

# Automating Belief Revision for AgentSpeak

Natasha Alechina<sup>1</sup>, Rafael H. Bordini<sup>2</sup>, Jomi F. Hübner<sup>3</sup>,  
Mark Jago<sup>1</sup>, and Brian Logan<sup>1</sup>

<sup>1</sup> School of Computer Science  
University of Nottingham  
Nottingham, UK  
{nza,mtw,bsl}@cs.nott.ac.uk

<sup>2</sup> University of Durham  
Dept. of Computer Science  
Durham, UK  
r.bordini@durham.ac.uk

<sup>3</sup> Univ. Regional de Blumenau  
Dept. Sistemas e Computação  
Blumenau, SC, Brazil  
jomi@inf.furb.br

**Abstract.** The AgentSpeak agent-oriented programming language has recently been extended with various new features, such as speech-act based communication, internal belief additions, and support for reasoning with ontological knowledge, which imply the need for belief revision within an AgentSpeak agent. In this paper, we show how a polynomial-time belief-revision algorithm can be incorporated into the *Jason* AgentSpeak interpreter by making use of *Jason*'s language constructs and customisation features. This is one of the first attempts to include automatic belief revision within an interpreter for a practical agent programming language.

## 1 Introduction

After almost a decade of work on abstract programming languages for multi-agent systems, practical multi-agent platforms based on these languages are now beginning to emerge. On such language is AgentSpeak, and in particular its implementation in *Jason* [7]. AgentSpeak continues to evolve, and a number of AgentSpeak extensions have been reported in the literature and incorporated into *Jason*. Some of these new features, such as internal belief additions, speech-act based communication, and support for reasoning with ontological knowledge, have led to a greater need for *belief revision* as part of an AgentSpeak agent's reasoning cycle. However, in common with other mature agent-oriented programming languages [5], AgentSpeak does not currently provide automatic support for belief revision. While the current *Jason* implementation provides a simple form of belief *update*, which can be customised for particular applications, the problem of belief-base consistency has, so far, remained the responsibility of the programmer.

The lack of support for belief revision in practical agent programming languages is understandable, given that known belief revision algorithms have high computational complexity. However recent work by Alechina, Jago and Logan has changed this picture. In [2] they presented a polynomial time belief revision algorithm for resource bounded agents. The algorithm is theoretically well-motivated, in the sense of producing revisions that conform to a generally accepted set of postulates characterising *rational* belief revision. In this paper, we show how this work can be incorporated into the *Jason* AgentSpeak interpreter by making use of *Jason*'s language constructs and customisation features. In doing so, we also clarify the desired outcome of belief contraction in AgentSpeak from the logical point of view.

The problem of how to incorporate belief revision into a practical agent programming language has been largely ignored in the literature. There has been some initial work on belief revision in an *abstract* programming language, for example, in [24]. In [10], Clark and McCabe show how the  $\text{Go!}$  agent programming language can be extended to incorporate dependency (or reason) maintenance. In their approach, the removal of a belief  $B$  automatically results in  $B$  being removed from justifications of other beliefs, and if  $B$  was the only justification, then those beliefs are automatically removed. However, the removal of sufficient beliefs to prevent  $B$  being re-derived is left to the programmer. We believe our approach is one of the first attempts to include automatic *belief revision* within an interpreter for a practical agent programming language.

The remainder of the paper is organised as follows. In Sections 2 and 3 we give a brief overview of AgentSpeak programming and its implementation in *Jason*. In Section 4, we state our desiderata for belief revision in AgentSpeak, and in Section 5 we summarise the main points of the algorithm first presented in [2]. We then discuss the integration of the belief revision algorithm into *Jason* in Section 7, and in Section 8 we give a simple example which illustrates the importance of belief revision in practical programming of multi-agent systems. Finally, we conclude and outline directions for future work.

## 2 AgentSpeak

The AgentSpeak(L) programming language was introduced in [22]. It is based on logic programming and provides an elegant abstract framework for programming BDI agents. The BDI architecture is, in turn, the predominant approach to the implementation of *intelligent* or *rational* agents [27], and a number of commercial applications have been developed using this approach.

An AgentSpeak agent is defined by a set of ground (first-order) atomic formulae which comprise its *belief base*, and a set of plans which form its *plan library*. An AgentSpeak plan has a *head* which consists of a triggering event (specifying the events for which that plan is *relevant*), and a conjunction of belief literals representing a *context*. The conjunction of literals in the context must be a logical consequence of that agent's current beliefs if the plan is to be considered *applicable* when the triggering event happens (only applicable plans can be chosen for execution). A plan also has a *body*, which is a sequence of basic actions or (sub)goals that the agent has to achieve

(or test) when the plan is triggered. *Basic actions* represent the atomic operations the agent can perform to change the environment. Such actions are also written as atomic formulæ, but using a set of *action symbols* rather than predicate symbols. AgentSpeak distinguishes two types of *goals*: achievement goals and test goals. Achievement goals are formed by prefixing atomic formulæ with the ‘!’ operator, while test goals are prefixed with the ‘?’ operator. An *achievement goal* states that the agent wants to achieve a state of the world where the associated atomic formula is true. A *test goal* states that the agent wants to test whether the associated atomic formula is a logical consequence of its beliefs.

An AgentSpeak agent is a *reactive planning system*. A plan execution is triggered by the *addition* (‘+’) or *deletion* (‘-’) of beliefs due to perception of the environment, or to the addition or deletion of goals as a result of the execution of plans triggered by previous events.

A simple example of an AgentSpeak program for a Mars robot is given in Figure 1. The robot is instructed to be especially attentive to “green patches” on rocks it observes while roving on Mars. The AgentSpeak program consists of three plans. The first plan says that whenever the robot perceives a green patch on a certain rock (a belief addition), it should try and examine that particular rock. However this plan can only be used (i.e., it is only applicable) if the robot’s batteries are not too low. To examine the rock, the robot must retrieve, from its belief base, the coordinates it has associated with that rock (this is the reason for the test goal in the beginning of the plan’s body), then achieve the goal of traversing to those coordinates and, once there, examine the rock. Recall that each of the achievement goals will trigger the execution of some other plan.

```
+green_patch(Rock)
  : not battery_charge(low)
  <- ?location(Rock,Coordinates);
      !traverse(Coordinates);
      !examine(Rock).

+!traverse(Coords)
  : safe_path(Coords)
  <- move_towards(Coords).

+!traverse(Coords)
  : not safe_path(Coords)
  <- ...
```

**Fig. 1.** Examples of AgentSpeak Plans for a Mars Rover

The two other plans (note the last one is only an excerpt) provide alternative courses of action that the rover should take to achieve the goal of traversing towards some given coordinates. Which course of action is selected depends on its beliefs about the environment at the time the goal-addition event is handled. If the rover believes that there is a safe path in the direction to be traversed, then all it has to do is to take the

action of moving towards those coordinates (this is a basic action which allows the rover to effect changes in its environment, in this case physically moving itself). The alternative plan (not shown here in full) provides an alternative means for the agent to reach the rock when the direct path is unsafe.

### 3 Jason

The *Jason* interpreter implements the operational semantics of AgentSpeak as given, e.g., in [8]. *Jason*<sup>4</sup> is written in Java, and its IDE supports the development and execution of distributed multi-agent systems [6]. Some of the features<sup>5</sup> of *Jason* are:

- speech-act based inter-agent communication (and annotation of beliefs with information sources);
- annotations on plan labels, which can be used by elaborate (e.g., decision-theoretic) selection functions;
- the possibility to run a multi-agent system distributed over a network (using SACI or some other middleware);
- fully customisable (in Java) selection functions, trust functions, and overall agent architecture (perception, belief-revision, inter-agent communication, and acting);
- straightforward extensibility (and use of legacy code) by means of user-defined “internal actions”;
- clear notion of *multi-agent environments*, implemented in Java (this can be a simulation of a real environment, e.g., for testing purposes before the system is actually deployed).

#### 3.1 Extensions to AgentSpeak

Recent work appearing in the literature has made important additions to AgentSpeak, which have also been (or are in the process of being) implemented in *Jason*. Below we briefly discuss some of these features, focusing on those that have particular implications for belief revision.

*Belief additions* This is one of the earliest extensions of the AgentSpeak language, and one of the most important from the point of view of belief revision. Experience with AgentSpeak has shown that the execution of some plans could be greatly facilitated by allowing a plan instance being executed to add derived beliefs to the agent’s belief base. A formula such as  $+bl$  in the body of a plan, has the effect of adding the belief literal  $bl$  to the belief base. Together with the ability to exchange beliefs and plans with other agents (see below), such derived beliefs can result in the agent’s belief base becoming inconsistent (i.e., both  $b$  and  $\sim b$  are in the belief base, for some belief  $b$ )<sup>6</sup>. Unless the programmer intends to make use of paraconsistency, this is clearly undesirable, yet it is *not* checked or handled automatically in the current version of *Jason*.

<sup>4</sup> *Jason* is *Open Source* (GNU LGPL) and available from <http://jason.sf.net>.

<sup>5</sup> As of this writing, the current version of *Jason* is v0.8.

<sup>6</sup> The ‘ $\sim$ ’ operator denotes strong negation in *Jason*.

*Speech-act based communication and plan exchange* Another important addition, first proposed in [17], is the extension of the AgentSpeak operational semantics to allow speech-act based communication among AgentSpeak agents. That work gave semantics to the change in the mental attitudes of AgentSpeak agents when receiving messages from other agents (using a speech-act based language). This involves not only changes in beliefs and goals, but also the plans used by the agent. The latter allows agents to exchange know-how with other agents in the form of plans for dealing with specific events [3]. The intuitive idea is that if one does not know how to do something, one should ask someone who does. To implement this idea of *cooperation* through plan exchange between agents, it was necessary to enable agents to retrieve plans for a given triggering event for which the agent has no applicable plan, and also to annotate plans with *access specifiers* (e.g., to prevent private plans being accessed by other agents) or with indications of what the agent should do with the retrieved plan once it has been used for a particular event (e.g., discard it, or keep it in the plan library for future reference). However the ability to exchange beliefs and plans with other agents increases the chances of the agent's belief base becoming inconsistent.

*Prolog-like Rules* An extension currently being implemented in *Jason* is that the belief base will no longer be just a set of ground literals, but will also have Prolog-like rules. This is similar to the 3APL implementation [11], where the belief base is effectively a Prolog program. The lack of such rules forced AgentSpeak programmers to use plans for deriving useful information, and the consequence of this was that programs were less clear because theoretical and practical reasoning were mixed together. With this addition, we now have a clear separation between rules that allow agents to derive conclusions that follow from their current belief state and plans which allow agents to decide how to act. However, for uniformity, we do not use the Prolog syntax for the body of rules; instead, we use the same syntax as used in the context of AgentSpeak plans (with the same expressive power).

*Ontological reasoning* Ontologies are presently being used in various agent-based applications (see, e.g., [9]). In [18], the AgentSpeak-DL extension of AgentSpeak was proposed which aimed at incorporating ontological reasoning within an AgentSpeak interpreter. The language was extended so that the belief base can include Description Logic [4] operators. In addition to the usual ABox (factual knowledge in the form of ground atomic formulæ), the belief base can also have a TBox (containing definitions of complex concepts and relationships between them). This results in a number of changes in the interpretation of AgentSpeak programs: (i) queries to the belief base are more expressive as their results do not depend only on explicit knowledge but can also be inferred from the ontology; (ii) the notion of belief update must be refined so that following the addition of a property of an individual, the resulting belief base is consistent with the agent's concept descriptions; (iii) the search for a plan (in the agent's plan library) that is relevant for dealing with a particular event is more flexible as this is not based solely on unification, but also on the subsumption relation between concepts; and (iv) agents may share knowledge by using web ontology languages such as OWL.

The issue of belief revision is clearly important in the context of ontological reasoning, and this is another motivation for the work presented here. In particular, consider item (ii) above. For example, if the belief base contains both  $unstable(p)$  and  $\neg dangerous(p)$ , and the TBox contains  $unstable \sqsubseteq dangerous$  then the agent’s belief state is inconsistent. However, simply ignoring the belief addition  $unstable(p)$  since it would cause an inconsistency in the belief base, or always forcing such additions and removing the contradicting belief instead, are both clearly unacceptable.

*Belief annotations* Another important change in the version of AgentSpeak interpreted by **Jason** is that atomic formulæ now can have “annotations”. An annotation is a list of terms enclosed in square brackets immediately following a predicate. For example, the annotated belief “`green_patch(r1)[doc(0.9)]`” could be used by a programmer to represent the fact that rock `r1` is believed to have a green patch in it, and this is believed with a degree of certainty (`doc`) of 0.9. Within the belief base, an important use of annotations is to record the sources of information for a particular belief. A (pre-defined) term `source(s)` is provided for that purpose, where `s` can be an agent’s name (to denote the agent that has communicated that information), or two special atoms, `percept` and `self`, which denote, respectively, that a belief arose from perception of the environment, or from the agent explicitly adding a belief to its own belief base as a result of executing a plan. The initial beliefs that are part of the source code of an AgentSpeak agent are assumed to be internal beliefs (i.e., as if they had a `[source(self)]` annotation), unless the belief has any source explicit annotation given by the user. For more on the annotation of sources of information for beliefs, see [17].

As will be seen below, annotations can be used to support context sensitive belief revision, where beliefs of a particular type or from a particular source are preferred to others when an inconsistency arises.

### 3.2 Belief Update in Jason

Users can customise certain aspects of the (practical) reasoning of a **Jason** agent by overriding methods of the `Agent` class. This includes, for example, the three user-defined selection functions that are required by an AgentSpeak interpreter. One of the methods of the `Agent` class that can be overridden, which is of interest here, is the `brf` method. This represents the *belief revision function* commonly found in agent architectures (although the Agents literature often assumes that this function is used mainly for belief update, rather than belief revision).

In the current version of **Jason**, the `brf` method takes a list of additions to the belief base, and is used for both belief *update* and belief revision. Belief update following perception of the environment results in a call to `brf` with literals in the list of additions representing the percepts<sup>7</sup>. It is assumed that all perceptible properties are included in the list of additions: all current beliefs no longer within the list of percepts are deleted from the belief base, and all percepts not currently in the belief base are added to it. In

<sup>7</sup> The fact that a literal is a percept rather than other forms of information is explicitly stated in the annotations: all percepts have a `source(percept)` annotation.

regards to other changes in the belief base, the default brf method in *Jason* simply adds to the belief base any belief addition executed within a plan, as well as any information from permitted sources; the source is annotated on the belief added to belief base, so that consideration of the degree of trust in any particular belief can be taken by the programmer. At present, belief additions (from whatever source) are *not* checked for consistency, with the result that the belief base can become inconsistent, unless sufficient care is taken by the programmer.

## 4 Requirements for Belief Revision in AgentSpeak

We have two main objectives in adding belief revision to AgentSpeak. First the belief revision algorithm should be theoretically well-motivated, in the sense of producing revisions which conform to a generally accepted set of postulates characterising *rational* belief revision. Second, we want the resulting language to be practical, which means that the belief revision algorithm must be efficient. Our approach draws on recent work [2] on efficient (polynomial-time) belief revision algorithms which satisfy the well-known AGM postulates [1] characterising rational belief revision and contraction.

The theory of belief revision as developed by Alchourron, Gärdenfors, and Makinson in [13, 1, 14] models belief change of an idealised rational reasoner. The reasoner's beliefs are represented by a potentially infinite set of beliefs closed under logical consequence. When new information becomes available, the reasoner must modify its belief set to incorporate it. The AGM theory defines three operators on belief sets: expansion, contraction, and revision. *Expansion*, denoted  $K + A$ , simply adds a new belief  $A$  to  $K$  and the resulting set is closed under logical consequence. *Contraction*, denoted by  $K \dot{-} A$ , removes a belief  $A$  from the belief set and modifies  $K$  so that it no longer entails  $A$ . *Revision*, denoted  $K \dot{+} A$ , is the same as expansion if  $A$  is consistent with the current belief set, otherwise it minimally modifies  $K$  to make it consistent with  $A$ , before adding  $A$ .

Contraction and revision cannot be defined uniquely, since in general there is no unique maximal set  $K' \subset K$  which does not imply  $A$ . Instead, the set of "rational" contraction and revision operators is characterised by the AGM postulates [1]. The basic AGM postulates for contraction are:

- (K-1)  $K \dot{-} A = Cn(K \dot{-} A)$  (closure)
- (K-2)  $K \dot{-} A \subseteq K$  (inclusion)
- (K-3) If  $A \notin K$ , then  $K \dot{-} A = K$  (vacuity)
- (K-4) If  $\text{not} \vdash A$ , then  $A \notin K \dot{-} A$  (success)
- (K-5) If  $A \in K$ , then  $K \subseteq (K \dot{-} A) + A$  (recovery)
- (K-6) If  $Cn(A) = Cn(B)$ , then  $K \dot{-} A = K \dot{-} B$  (equivalence)

where  $Cn(K)$  denotes closure of  $K$  under logical consequence.

AGM style belief revision is sometimes referred to as *coherence* approach to belief revision, because it is based on the ideas of coherence and informational economy. It requires that the changes to the agent's belief state caused by a revision be as small as possible. In particular, if the agent has to give up a belief in  $A$ , it does not have to

give up believing in things for which  $A$  was the sole justification, so long as they are consistent with the remaining beliefs.

While AGM belief revision provides an appealing definition of rational belief revision, it is generally considered to apply only to idealised agents, because of the assumption that the set of beliefs is closed under logical consequence. To model AI agents, an approach called belief base revision has been proposed (see for example [16, 19, 25, 23]). A belief base is a finite representation of a belief set. Revision and contraction operations can be defined on belief bases instead of on logically closed belief sets. However the complexity of these operations ranges from NP-complete (full meet revision) to low in the polynomial hierarchy (computable using a polynomial number of calls to an NP oracle which checks satisfiability of a set of formulæ) [21]. The reason for the high complexity is the need to check for classical consistency while performing the operations. One way around this is to weaken the language and the logic used by the agent so that the consistency check is no longer an expensive operation (as suggested in [20]). This is also the approach taken in [2] and adopted here.

Another strand of theoretical work in belief revision is the *foundational*, or *reason-maintenance* approach to belief revision. Reason-maintenance style belief revision is concerned with tracking dependencies between beliefs. Each belief has a set of justifications, and the reasons for holding a belief can be traced back through these justifications to a set of foundational beliefs. When a belief must be given up, sufficient foundational beliefs have to be withdrawn to render the belief underivable. Moreover, if all the justifications for a belief are withdrawn, then that belief itself should no longer be held. Most implementations of reason-maintenance style belief revision are incomplete in the logical sense, but tractable.

In the next section we present an approach to belief revision and contraction for resource-bounded agents which allows for both AGM and reason-maintenance style belief revision. AGM-style contraction by  $A$  removes beliefs which imply  $A$ , but does not remove beliefs for which  $A$  is the sole justification. In contrast, reason-maintenance style contraction in addition removes beliefs for which  $A$  is the sole justification. Both AGM and reason-maintenance contraction have the same polynomial-time complexity and satisfy the AGM postulates with the exception of the recovery postulate (K-5).

## 5 The Belief Revision Algorithm

In this section we briefly describe the contraction algorithm introduced in [2]. We define AGM-style contraction by a literal  $A$  as the removal of  $A$  and sufficient literals from the agent's belief base so that  $A$  is no longer a consequence of the beliefs in the belief base. We explain in more detail in Section 6 what we mean by "consequence"; for the moment, we assume that a belief is derivable if it has been asserted using the agent's plans or has been inferred using ontological definitions and the literals in the agent's belief base.

The inferential relationships between the beliefs in the agent's belief base can be represented as a directed graph, where the nodes are beliefs and *justifications*. A justification consists of a belief literal and a *support list* containing the beliefs which were used to derive that literal, for example:  $(A, [B, C])$ , where  $A$  was derived from  $B$  and

$C$ . If  $A$  has been derived in several different ways, for example, from  $B, C$  and from  $D$  (where  $B, C$  and  $D$  are in the belief base), the graph contains several justifications for  $A$ , for example  $(A, [B, C])$  and  $(A, [D])$ . We say a belief is *independent* if it has at least one non-inferential justification, e.g., beliefs acquired by perception or communicated, and the literals in the belief base when the agent starts. Non-inferential justifications are of the form  $(D, [])$ , i.e., the support list is empty. In the graph, each justification has one outgoing edge to the belief it is a justification for, and an incoming edge from each belief in its support list. We assume that each support list  $s$  has a designated *least preferred* member  $w(s)$ . Intuitively, this is a belief which is not preferred to any other belief in the support list, and which we would be prepared to discard first, if we have to give up one of the beliefs in the list. We discuss possible preference orderings and their computation in the next section. We assume that we have constant time access to  $w(s)$ .

The algorithm to contract by a belief  $A$  is as follows:

```

For each of A's outgoing edges
  to a justification (C, s),
    remove (C,s) from the graph.

```

```

For each of A's incoming edges
  from a justification (A, s),
    if s is empty:
      remove (A, s);
    else:
      contract by w(s);

```

Remove  $A$ .

To implement reason-maintenance style contraction, we also recursively remove beliefs which have no incoming edges.

The algorithm runs in time  $O(kr + n)$ , where  $k$  the maximal number of beliefs in any support list,  $r$  is the number of plans, and  $n$  the number of literals in the belief base [2]. Indeed, the upper bound on the number of steps required to remove justifications corresponding to plan instances is  $r(k + 1)$  (one constant time operation for each belief in the context of the plan and one for the belief asserted by the plan). Removing all justifications corresponding to foundational beliefs costs  $n$  steps. The last step in the contraction algorithm (removing a belief) is executed at most  $n$  times. Reason-maintenance style contraction adds one extra traversal of the justification graph, but has the same complexity.

In [2], it was shown that the contraction operator defined by the algorithm above satisfies (K-1)–(K-4) and (K-6); (K-5) is not satisfied. To give an example, suppose  $B$  can be derived using  $A$  (but not vice versa). If we contract by  $B$ , then  $A$  is also removed from the belief base. When we expand by  $B$ ,  $A$  is not restored to the belief base.

## 5.1 Preferred Contractions

In general, an agent will prefer some contractions to others. In this section we focus on contractions based on preference orders over individual beliefs, e.g., degree of belief or

commitment to beliefs. We assume that an agent associates an *a priori* quality with each non-inferential justification for its independent beliefs. For example, communicated information may be assigned a degree of reliability by its recipient which depends on the degree of reliability of the speaker (i.e., the speaker’s reputation), percepts may be assumed to be more reliable than communicated information, and so on.

For simplicity, we assume that quality of a justification is represented by non-negative integers in the range  $0, \dots, m$ , where  $m$  is the maximum size of the belief base. A value of 0 means the lowest quality and  $m$  means highest quality. We take the preference of a literal  $A$ ,  $p(A)$ , to be that of its highest quality justification:

$$p(A) = \max\{qual(j_0), \dots, qual(j_n)\},$$

where  $j_0, \dots, j_n$  are all the justifications for  $A$ , and we define the quality of an inferential justification to be that of the least preferred belief in its support<sup>8</sup>:

$$qual(j) = \min\{p(A) : A \in \text{support of } j\}.$$

This is similar to ideas in argumentation theory: an argument is only as good as its weakest link, yet a conclusion is at least as good as the best argument for it. This approach is also related to Williams “partial entrenchment ranking” [26] which assumes that the entrenchment of any sentence is the maximal quality of a set of sentences implying it, where the quality of a set is equal to the minimal entrenchment of its members. While this approach is intuitively appealing, nothing hangs on it, in the sense that any preference order can be used to define a contraction operation, and the resulting operation will satisfy the postulates.

To perform a preferred contraction, we preface the contraction algorithm given above with a step which computes the preference of each literal in the belief base, and for each justification, finds the position of a least preferred member of the support list. The preference computation algorithm can be found in [2]. We then simply run the contraction algorithm to recursively delete the weakest member of each support in the dependency graph of  $A$ .

We define the *worth* of a set of literals  $\Gamma$  as  $worth(\Gamma) = \max\{p(A) : A \in \Gamma\}$ . In [2], it was shown that the contraction algorithm removes the set of literals with the least worth. More precisely:

**Proposition 1.** *If contraction of the set of literals in the belief base  $K$  by  $A$  resulted in removal of the set of literals  $\Gamma$ , then for any other set of literals  $\Gamma'$  such that  $K - \Gamma'$  does not imply  $A$ ,  $worth(\Gamma) \leq worth(\Gamma')$ .*

The proof is given in [2]. Computing preferred contractions involves only modest computational overhead. The total cost of computing the preference of all literals in the belief base is  $O(n \log n + kr)$ , where  $n$  the number of literals in the belief base,  $k$  is the maximal number of beliefs in any support list, and  $r$  the number of plans. As the contraction algorithm is unchanged, this is also the additional cost of computing a preferred contraction. Computing the most preferred contraction can therefore be performed in time linear in  $kr + n$ .

<sup>8</sup> Literals with no supports (as opposed to an empty support) are viewed as having an empty support of the lowest quality.

## 6 Belief Revision in AgentSpeak

The belief revision algorithm presented above was developed for rule based agents, where rules could easily be interpreted as corresponding to logical implications. However the execution of AgentSpeak plans cannot always be interpreted as corresponding to logical derivations. In this section we show how the algorithm can be applied in the context of AgentSpeak and briefly discuss the notion of consistency which our contraction algorithm maintains between the beliefs of an AgentSpeak agent.

The original version of the algorithm presented in [2] assumed a forward-chaining rule-based agent, with rules of the form  $A_1, \dots, A_n \rightarrow B$ , which fires its rules to quiescence (i.e., until no more rules are applicable). To define, for those agents, what it means for the belief base to be consistent and what it means for a belief to be derivable, we interpret rules as Horn clauses and literals as atomic formulæ of predicate logic, and postulate that the agent “reasons” using a single inference rule of *generalised modus ponens*:

$$\frac{\delta(A_1), \dots, \delta(A_n), \quad \forall \bar{x}(A_1 \wedge \dots \wedge A_n \rightarrow B)}{\delta(B)}$$

where  $\delta$  is a substitution function which replaces all free variables of a formula with constants. In [2], the resulting logic is called  $W$ . Clearly, the language of  $W$  is weaker than the language of full classical logic (e.g., it does not have disjunctions). The deductive power of the logic is also weaker; for example, from  $A \rightarrow B$  and  $\neg A \rightarrow B$  the agent cannot derive  $B$ , as it would have been possible in classical logic. It was shown in [2] that the beliefs of a quiescent forward-chaining agent are closed with respect to consequence in  $W$ , and that the contraction algorithm restores consistency in the sense of  $W$ ; that is, after contraction by  $A$  the agent can no longer derive  $A$  in  $W$  from its beliefs.

Since not all rule-based agents fire their rules to quiescence, it was further analysed in [2] the case of belief contraction for non-quiescent rule-based agents. In that case, the agent’s beliefs at each point in time are closed with respect to the set of rule instances *which have been applied by the agent up to that point in time*. Even this is not exactly true if the agent uses refractory rule firing, that is, each rule instance is only fired once. This means that if  $A(x)$  implies  $B(x)$ ,  $A(a)$  is in the belief base,  $B(a)$  is derived and then removed,  $B(a)$  can not be re-derived again using the same rule. However in this paper we ignore refractory rule firing and take “derivable” to mean “derivable using previously fired rule instances”. In effect, we over-approximate the set of literals the agent can derive with respect to refractory rule firing.

As noted above, the execution of AgentSpeak plans cannot always be interpreted as corresponding to logical derivations. We therefore assume that, in future versions of **Jason**, plans that can be used to derive a new belief on the basis of currently held beliefs will be annotated by the programmer to indicate that they are relevant for belief revision. We call such plans *declarative rules*; these also include the Prolog-like rules that will be available in **Jason** as explained in Section 3. A declarative-rule plan of the form “ $te : l_1 \& \dots \& l_n \leftarrow bd$ ”, where  $te$  is a triggering event and  $bd$  a plan body starting with belief addition  $+bl$ , is interpreted as an implication  $l_1, \dots, l_n \rightarrow bl$ ; if the triggering event,  $te$ , is itself a belief addition, it is included in the antecedent as well:

$te, l_1, \dots, l_n \rightarrow bl$ . When declarative rules or ontological reasoning are used to check if a plan is applicable, e.g., if a plan with literals  $l_1, \dots, l_n$  in its context is applicable not because  $l_1, \dots, l_n$  are in the belief base, but because other literals  $m_1, \dots, m_n$  are, and the agent’s declarative rules or ontological reasoning entails  $m_i \sqsubseteq l_i$  for each  $i$ , we interpret this as a plan instance with  $m_1, \dots, m_n$  in its context. A belief is then derivable if it is derivable from the belief base using the plan instances used so far, with the results of the declarative rules or ontological reasoning substituted in the actual plan instances; so instead of, e.g.,  $l_1, \dots, l_n \rightarrow bl$  in the implications, we have  $m_1, \dots, m_n \rightarrow bl$  where  $m_1, \dots, m_n$  are the belief literals in the belief base which were actually used for the plan to be considered applicable, given the agent’s declarative or ontological rules.

We say that a belief base of an AgentSpeak agent is consistent if a contradiction is not derivable by generalised modus ponens from the contents of the belief base and the agent’s plans (represented as implications) which were used in the agent’s execution so far. In particular, after contracting by  $bl$ ,  $bl$  is not re-derivable from the corresponding set of implications and atomic beliefs in logic  $W$ .

## 7 Implementation of Belief Revision in *Jason*

Future versions of *Jason* will incorporate an implementation of the belief revision algorithm described above. The belief revision functionality will be split between the existing `brf` and a new `buf` responsible for belief update. Perception of the environment will be followed by a call to `buf` with the literals in the list of additions representing the list of percepts. As in the current version of *Jason*, it is assumed that all perceptible properties are included in the list of additions: all current beliefs no longer within the list of percepts are deleted from the belief base, and all percepts not currently in the belief base are added to it. All other additions to the belief base are handled by `brf`. Belief additions in plans annotated for belief revision result in a call to `brf` with the new belief as argument. If the new belief is inconsistent with the current belief base, `brf` may discard it or may delete some other belief(s) to allow the new belief to be consistently added to the belief base. Which beliefs are actually deleted is determined by a user-specified preference order (see below).

The directed graph used by the belief revision algorithm is implemented in terms of two lists for each belief: a “dependencies list” (the literals that allowed the derivation of the belief literal in question), and a “justifies list” (which other beliefs the literal in question justifies, i.e., it appears in their dependencies list). The new `brf` method maintains two special annotations for each belief in the belief base “`dep([...])`, `just([...])`” which record the dependencies and justifies lists for the belief respectively. The literals to populate these two lists are retrieved from the intention that generated the belief change. For example, if the top of the intention structure that generated a belief change  $+bl$  has a plan instance of the form “ $te : l_1 \& \dots \& l_n \leftarrow bd$ ”, where  $te$  is a triggering event and  $bd$  a plan body in which  $+bl$  is the first action, the support list of the justification is simply the (ground) literals from the plan context, “ $[l_1, \dots, l_n]$ ”. If the triggering event,  $te$ , is itself a belief (addition), the literal in  $te$  is included together with the context literals in the support list as noted above. If a belief was added as a result of a plan being applicable using declarative

rules or ontological reasoning — e.g., literal  $l_1$  is not in the belief base, but some other literal  $l$  is, and  $l \sqsubseteq l_1$  is an instance of an declarative or ontology rule — then  $l$  will be in the support list for  $bl$  instead of  $l_1$ . For each literal in the support list  $l_1, \dots, l_n$  we add the justification to the literal’s “justifies” list. We also record the time at which the justification was added to the relevant list.

In addition to the “dependencies” and “justifies” lists, the belief revision algorithm also requires the definition of a partial order relation specifying contraction preference. To allow for user customisation, this is defined as a separate method that can also be overridden. The default definition of this method gives preference to perceived information over communicated information (as also happens in [24]), and in case of information from sources of similar reliability, it gives preference to newer information over older information (this is why the time when a justification was inserted is also added as an annotation).

During belief update and revision, the “dependencies” and “justifies” lists are updated to reflect the beliefs (and their justifications) which have been removed from the agent’s belief base. Deletion of beliefs as a result of belief update results in the appropriate changes to the justification structure and belief base: with AGM-style revision, the deleted belief is removed from the “dependencies” lists of all other beliefs in the belief base; with reason-maintenance style revision, any beliefs in whose “justifies” list a deleted belief appears are also deleted. The `buf` method also removes any justification which has a deleted perceptual belief in its support list, and marks the justified belief as a self-supporting (independent) belief. This additional bookkeeping reflects the special status of perceptual beliefs: perceptual beliefs can trigger or justify the addition of other beliefs, but their deletion is not in itself a reason for removing a derived belief. When an inconsistency is detected during belief revision, `brf` determines which of the two beliefs is less preferred, and then (recursively) deletes the least preferred belief in the “dependencies” list of the contracted belief. With reason-maintenance style revision, any beliefs in whose “justifies” list a deleted belief appears are also deleted.

The implementation described above is conservative in revising only the agent’s belief state. The agent’s plans are considered part of the agent’s program and are not revised (though revising plans, in particular those received from other agents, would be an interesting extension). Similarly, when revising beliefs derived using ontological rules, we assume the ontology used by the agent to be immutable and consistent, and that it is consistent with every other ontology it references. Moreover, *intention* revision remains the responsibility of the programmer. Changes in the agent’s intentions following the removal of beliefs to restore consistency, or changes in beliefs as a consequence of intention reconsideration, must be programmed using the appropriate *Jason* mechanisms. All belief changes, regardless of whether they are internal, communicated, or perceived can lead to the execution of a plan which could be used, for example, to drop an intention. If the belief revision algorithm has to remove any beliefs to ensure consistency, this will also generate the appropriate (belief-deletion) internal events, which in turn can trigger the execution of a such plans to revise the agent’s intentions.

## 8 An Example

To illustrate the importance of belief revision in the context of AgentSpeak (or, more generally, an agent programming language), we present a simple scenario of an agent that buys stocks from the stock market. The agent receives financial information (or guesses) from other agents, some of which can be trusted (or are currently considered trustworthy), and it also has access to Web Services which filter relevant newspaper stories and provide symbolic versions of such news for stock market agents. As these web services are authenticated, this corresponds to actual perception of the “environment”.

Suppose our agent receives a message  $\langle ag1, tell, salesUp(c1) \rangle$  and its plan library has the following plan:

```
+salesUp(C)[source(A)]
  : wellManaged(C) & trust(A)
  <- +goodToBuy(C).
```

When the plan is executed, the `brf()` method will then add `goodToBuy(c1)[source(ag1)]` to the belief base with `[salesUp(c1), wellManaged(c1), trust(ag1)]` in its “dependencies” list, and `goodToBuy(c1)` is added to the “justifies” lists of the beliefs `salesUp(c1)`, `wellManaged(c1)`, and `trust(ag1)`. In the context of the overall agent program, the idea is that if the agent ever comes to have the goal of buying stocks, it can make use of beliefs such as `goodToBuy`, together with various other conditions, to decide which specific stocks to buy.

Now assume that, from the financial news web service, the agent acquires the belief `stocks(c2,10)[source(percept)]`, which means that company `c2`'s stocks are up by 10 points, and the agent also believes that `rival(c2,c1)` (i.e., that companies `c2` and `c1` are competitors), so that increase in the stocks of one of them tends to lead to a decrease in the other's stocks. Assume further that the agent happens to have the following plan:

```
+stocks(C,P)
  : P > 5 & rival(C,R)
  <- +~goodToBuy(R).
```

When the plan is executed, the attempt to simply add `~goodToBuy(c1)` to the belief base would not be carried out because it would result in an inconsistent belief state. With the default contraction preference relation, it is not difficult to see that, in this instance, the algorithm would contract by `goodToBuy(c1)` because its support is based on communicated information which is less reliable than the observed information from which `~goodToBuy(c1)` was derived.

As can be seen, the belief revision algorithm takes care of ensuring that inconsistencies, such as `goodToBuy(c1)` and `~goodToBuy(c1)` being believed simultaneously, never occur in the belief base. Moreover, the data structures used by the algorithm (the dependencies and justifications lists) allow it to automatically revise the belief base in ways that previously would have required significant programming efforts. For example, suppose the agent receives news that a crooked CEO has just been fired from `c1`. The agent is likely to have a plan to update its beliefs about `c1` being

well managed as a consequence of such new information about the CEO. If the user has chosen the *reason-maintenance style* of the algorithm, and there is no other justification for `goodToBuy`, then the algorithm would remove not only the `wellManaged(c1)` belief, but also the `goodToBuy(c1)` belief because the latter depends on the former. Similarly if for some reason the agent later finds out that `ag1` is not trustworthy after all.

However, if the user opts for AGM-style revision, removing `wellManaged` would *not* remove `goodToBuy`. Although in this example the reason-maintenance style is arguably more appropriate, in other applications the coherence style might be more useful. In either case, it is clear that without the use of automatic belief revision, it would be very difficult for a programmer to ensure that revision occurs appropriately in all situations. The programmer would either have to develop an application-specific `brf` method, or else write specific plans to handle *all possible belief change events* that might affect any derivations the agent can make.

## 9 Conclusions and Future Work

As multi-agent programming languages become richer, it becomes harder for programmers to ensure that the belief states of agents developed using these languages are kept consistent. In this paper we briefly summarised the rationale for including automatic belief revision in an agent programming language. Using the AgentSpeak programming language as an example, we showed how a number of features recently added to the language have dramatically increased the need for automatic belief revision. We motivated the choice of the polynomial-time belief revision algorithm presented in [2], and described its integration into the *Jason* AgentSpeak interpreter. We also gave a simple example which illustrates the usefulness of such an automatic belief revision mechanism in a practical multi-agent system scenario, and sketched how it can significantly reduce the programming efforts required. We believe that other agent-oriented programming languages and their platforms [5], which currently push responsibility for maintaining a consistent belief state onto programmers, can also benefit from our approach.

A limitation of the work presented here is that we only consider plans which correspond to classical implication; in particular, we don't consider negation as failure in plan contexts. Belief revision in the presence of negation as failure would be an interesting problem to consider, since it corresponds to belief revision in default logic. Another interesting question to be considered in further work is the need for belief revision as a consequence of intention reconsideration. On the more practical side, we plan to develop large-scale agent applications to assess the performance of *Jason* with belief revision.

## Acknowledgements

Rafael Bordini gratefully acknowledges the support of The Nuffield Foundation (grant number NAL/01065/G).

## References

1. C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:510–530, 1985.
2. N. Alechina, M. Jago, and B. Logan. Resource-bounded belief revision and contraction. In *Proceedings of the 3rd International Workshop on Declarative Agent Languages and Technologies (DALT 2005)*, number 3904 in LNCS, pages 141–154, Utrecht, the Netherlands, July 2005. Springer-Verlag.
3. D. Ancona, V. Mascardi, J. F. Hübner, and R. H. Bordini. Coo-AgentSpeak: Cooperation in AgentSpeak through plan exchange. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2004)*, New York, NY, 19–23 July, pages 698–705, New York, NY, 2004. ACM Press.
4. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *Handbook of Description Logics*. Cambridge University Press, Cambridge, 2003.
5. R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*. Number 15 in Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, 2005.
6. R. H. Bordini, J. F. Hübner, et al. *Jason: A Java-based agentSpeak interpreter used with saci for multi-agent distribution over the net*, manual, release version 0.7 edition, August 2005. <http://jason.sourceforge.net/>.
7. R. H. Bordini, J. F. Hübner, and R. Vieira. *Jason* and the Golden Fleece of agent-oriented programming. In Bordini et al. [5], chapter 1.
8. R. H. Bordini and Á. F. Moreira. Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L). *Annals of Mathematics and Artificial Intelligence*, 42(1–3):197–226, Sept. 2004. Special Issue on Computational Logic in Multi-Agent Systems.
9. H. Chen, T. Finin, and A. Joshi. The SOUPA Ontology for Pervasive Computing. In V. T. et al, editor, *Ontologies for Agents: Theory and Experiences*, pages 233–258. BirkHauser, 2005.
10. K. L. Clark and F. G. McCabe. Ontology schema for an agent belief store. *IJGIS*, 2006. To appear.
11. M. Dastani, M. B. van Riemsdijk, and J.-J. C. Meyer. Programming multi-agent systems in 3APL. In Bordini et al. [5], chapter 2.
12. J. de Bruijn, A. Polleres, and D. Fensel. Owl lite<sup>-</sup>. working draft, wsmml deliverable d20 v0.1, WSML, 18th July 2004. <http://www.wsmo.org/2004/d20/v0.1/20040629/>.
13. P. Gärdenfors. Conditionals and changes of belief. In I. Niiniluoto and R. Tuomela, editors, *The Logic and Epistemology of Scientific Change*, pages 381–404. North Holland, 1978.
14. P. Gärdenfors. *Knowledge in Flux: Modelling the Dynamics of Epistemic States*. The MIT Press, Cambridge, Mass., 1988.
15. I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004*, pages 723–731. ACM, 2004.
16. D. Makinson. How to give it up: A survey of some formal aspects of the logic of theory change. *Synthese*, 62:347–363, 1985.

17. Á. F. Moreira, R. Vieira, and R. H. Bordini. Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In J. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Declarative Agent Languages and Technologies, Proc. of the First Int. Workshop (DALT-03), held with AAMAS-03, 15 July, 2003, Melbourne, Australia*, number 2990 in LNAI, pages 135–154, Berlin, 2004. Springer-Verlag.
18. A. F. Moreira, R. Vieira, R. H. Bordini, and J. Hübner. Agent-oriented programming with underlying ontological reasoning. In *Proceedings of the 3rd International Workshop on Declarative Agent Languages and Technologies (DALT 2005)*, number 3904 in LNCS, pages 155–170, Utrecht, the Netherlands, July 2005. Springer-Verlag.
19. B. Nebel. A knowledge level analysis of belief revision. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the First International Conference*, pages 301–311, San Mateo, 1989. Morgan Kaufmann.
20. B. Nebel. Syntax-based approaches to belief revision. In P. Gärdenfors, editor, *Belief Revision*, volume 29, pages 52–88. Cambridge University Press, Cambridge, UK, 1992.
21. B. Nebel. Base revision operations and schemes: Representation, semantics and complexity. In A. G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94)*, pages 341–345, Amsterdam, The Netherlands, August 1994. John Wiley and Sons.
22. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde and J. Perram, editors, *Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), 22–25 January, Eindhoven, The Netherlands*, number 1038 in Lecture Notes in Artificial Intelligence, pages 42–55, London, 1996. Springer-Verlag.
23. H. Rott. “Just Because”: Taking belief bases seriously. In S. R. Buss, P. Hájaek, and P. Pudlák, editors, *Logic Colloquium '98—Proceedings of the 1998 ASL European Summer Meeting*, volume 13 of *Lecture Notes in Logic*, pages 387–408. Association for Symbolic Logic, 1998.
24. R. M. van Eijk, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. Information-passing and belief revision in multi-agent systems. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V — Agent Theories, Architectures, and Languages, 5th International Workshop, ATAL '98, Paris, France, July 4-7, 1998, Proceedings*, volume 1555 of LNCS, pages 29–45, Berlin, 1999. Springer-Verlag.
25. M.-A. Williams. Two operators for theory base change. In *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence*, pages 259–265. World Scientific, 1992.
26. M.-A. Williams. Iterated theory base change: A computational model. In *Proceedings of Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1541–1549, San Mateo, 1995. Morgan Kaufmann.
27. M. Wooldridge. *Reasoning about Rational Agents*. The MIT Press, Cambridge, MA, 2000.