Automating Belief Revision for AgentSpeak

Natasha Alechina[1], Rafael H. Bordini[2], Jomi F. Hübner[3],
Mark Jago[1], and Brian Logan[1]

[1] School of Computer Science
University of Nottingham
Nottingham, UK
{nza,mtw,bsl}@cs.nott.ac.uk

[2] University of Durham
Dept. of Computer Science
Durham, UK
r.bordini@durham.ac.uk

[3] 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. One example of a well-known agent language that has evolved to the point of being sufficiently practical for widespread use is AgentSpeak, and in particular its implementation in ***Jason*** [7]. A number of extensions to AgentSpeak have been reported in the literature and incorporated into ***Jason***. Some of these new features, such as speech-act based communication, internal belief additions, and support for reasoning with ontological knowledge, have led to a greater need for *belief revision* as part of an agent's reasoning cycle. However, in common with other mature agent-oriented programming languages [5], ***Jason*** does not currently provide automatic support for belief revision. The current implementation provides a simple form of belief *update*, which can be customised for particular applications. However, 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 bounds. However recent work by Alechina et al. [2] has changed this picture. By making simplifying assumptions, which nevertheless are quite realistic for agent-oriented programming languages, they were able to produce a polynomial-time belief-revision algorithm, which is also theoretically well-motived, 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. This is one of the first attempts to include automatic belief revision within an interpreter for a practical agent programming language. Some initial considerations on belief revision in an *abstract* programming language appeared, for example, in [23]. In an approach similar to ours, [10] sketch how the `Go!` programming language can be extended with a consistency maintenance system which can be used by an agent whose beliefs are constrained by a formal ontology to decide which beliefs to remove in order to restore consistency.

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 introduced in [2]. We then discuss the integration of the belief revision algorithm into ***Jason*** in Section 6, while Section 7 gives a simple example which illustrates the importance of belief revision in practical programming of multi-agent systems. Finally, we discuss conclusions and future work.

2 AgentSpeak

The AgentSpeak(L) programming language was introduced in [21]. 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 [26], and a number of commercial applications have been developed using this approach.

An AgentSpeak agent is defined by a set of *beliefs* giving the initial state of the agent's *belief base*, which is a set of ground (first-order) atomic formulæ, 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 so as 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 an atomic formulæ prefixed 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 formulæ is true. A *test goal* states that the agent wants to test whether the associated atomic formulæ is (or can be unified with) one of its beliefs.

An AgentSpeak agent is a *reactive planning system*. Plans are 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, examining the rock. Recall that each of these 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 a 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) 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 in, e.g., [8]. ***Jason*** [4] is written in Java, and its IDE supports the development and execution of distributed multi-agent systems [6]. Some of the features 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*, which can be 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 One of the earliest extensions of the AgentSpeak language is one of the most important from the point of view of belief revision. From the initial work on AgentSpeak, experience showed that it was often the case 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 plans with other agents (see below), such derived beliefs can result in the agent's belief base becoming inconsistent (i.e., both b and **~**b are in the belief base, for some belief b)[5]. Unless the programmer deliberately intends to make use of paraconsistency, this is clearly undesirable, yet it is *not* currently checked or handled by ***Jason*** automatically.

Speech-act based communication and plan exchange Another important addition, first proposed in [16], is the extension of the AgentSpeak operational semantics to allow speech-act based communication among AgentSpeak agents. That work gave semantics

[4] ***Jason*** is *Open Source* (GNU LGPL) and is available from `http://jason.sourceforge.net`

[5] The '**~**' operator denotes strong negation in ***Jason***.

to the change in the mental attitudes of AgentSpeak agents when receiving messages from other agents (using a speech-act based language). This includes not only changes in beliefs and goals, but also the plans used by the agent. This 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. However, to systematise this idea, hence introducing the possibility of *cooperation* among agents, it was necessary not only the means for the retrieval of external plans for a given triggering event for which the agent has no applicable plan, but 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).

Ontological reasoning In [17], an 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; the extended language was called AgentSpeak-DL. 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 is refined so that a property about an individual can only be added if the resulting belief base is consistent with the concept description; (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 (e.g., item (ii) above), and this is another motivation for the work presented here. Further, ontologies are presently being used in various agent-based applications (see, e.g., [9]).

Although AgentSpeak-DL is not yet available in the latest release of ***Jason***, we briefly outline how our work on belief revision will combine with the ongoing implementation of AgentSpeak-DL. In ***Jason***, the abstract language presented in [17] will take the following more practical form. We will represent ontological knowledge in OWL Lite$^-$ [11], or in the form of Horn clauses. Interestingly, the OWL Lite$^-$ language was created precisely so that any ontology thus defined could be translated into Datalog, hence efficient query answering could be done based on logic programming techniques. Unlike the abstract language used in [17], definitions such as

$$presenter \equiv invitedSpeaker \sqcup paperPresenter.$$

are not allowed in the practical language to be used in this work (the best that we can do here are definitions such as $invitedSpeaker \sqsubseteq presenter$ and $paperPresenter \sqsubseteq presenter$). On the other hand, we will be able to express *ontology rules* [14] which are not expressible in description logic.

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, and 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 (this could be useful if the programmer wants the agent to have an initial belief as if it had been perceived from the environment, or as if it had been communicated by another agent). For more on the annotation of sources of information for beliefs, see [16].

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. 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 revision). To create a customised agent class which overrides the brf method (e.g., to include a more sophisticated algorithm than the standard one distributed with ***Jason***), the following method needs to be overridden[6]:

```
public class MyAgent extends Agent {

  public List[] brf(List adds, List dels) {
    // This function should revise the belief base
    // with the given literals to add and delete

    // In its return, List[0] has the list of actual
    // additions to the belief base, and List[1] has
    // the list of actual deletions; this is used to
    // generate the appropriate internal events
  }
}
```

[6] Note that the signature of the brf method as given below is different from what is currently available in ***Jason***, but this is how it will be in the next public release.

In the current ***Jason*** implementation, the brf method receives only a list of additions, and is used both for belief revision and belief update (i.e., perception of the environment is followed by a call to this method with literals representing the percepts[7]). For belief update following perception of the environment, 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. For belief revision, 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 trusted sources (note, however, that the source is annotated on the belief added to belief base, so in practice further consideration of the degree of trust in any 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 much care is taken by programmers.

4 Requirements for Belief Revision in AgentSpeak

We have two main objectives in our introduction of belief revision in AgentSpeak. First the algorithm should be theoretically well motived, 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 [12, 1, 13] 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 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]. Below, $Cn(K)$ denotes closure of K under logical consequence.

The basic AGM postulates for contraction are:

(K$\dot{-}$1) $K \dot{-} A = Cn(K \dot{-} A)$ (closure)
(K$\dot{-}$2) $K \dot{-} A \subseteq K$ (inclusion)

[7] 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.

(K$\dot{-}$3) If $A \notin K$, then $K \dot{-} A = K$ (vacuity)

(K$\dot{-}$4) If not $\vdash A$, then $A \notin K \dot{-} A$ (success)

(K$\dot{-}$5) If $A \in K$, then $K \subseteq (K \dot{-} A) + A$ (recovery)

(K$\dot{-}$6) If $Cn(A) = Cn(B)$, then $K \dot{-} A = K \dot{-} B$ (equivalence)

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.

AGM belief revision 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 [15, 18, 24, 22]). 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 formulas) [20]. 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 of the agent so that the consistency check is no longer an expensive operation (as suggested in [19]). This is also the approach taken in [2] and adopted here.

The 'language' of an AgentSpeak agent is weaker than the language of full classical logic (the belief base contains only literals) and the deductions the agent can make are limited to what can be expressed as plans (and, for example, ontology rules). We introduce belief revision operators in AgentSpeak which satisfy all but one of the AGM postulates (recovery is not satisfied), but the logical closure Cn in the postulates is interpreted as closure with respect to a logic which is weaker than full classical logic. This allows us to define theoretically sound, but efficient belief revision operations.

Another strand of theoretical work in belief revision is the *foundational*, or *reason-maintenance* style 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 both AGM and reason-maintenance style belief revision.

5 The Belief Revision Algorithm

In this section we briefly describe the linear-time contraction algorithm introduced in [2]. The algorithm defines resource-bounded 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 derivable.

Assume that the agent's belief base is a directed graph, where the nodes are beliefs and *justifications*. A justification consists of a belief and a *support list* containing the context (and possibly the triggering event) of the plan used to derive this belief, for example: $(A, [B, C])$, where A is a derived belief and it was asserted by a plan with context B and triggering belief addition C (or derived by an ontology rule $B, C \rightarrow A$). If A can be 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 $(A, [D])$. Foundational beliefs which were not derived, have a justification of the form $(D, [])$. 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 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 type contraction, we also remove beliefs which have no incoming edges.

In [2], it was shown that the contraction operator defined by the algorithm satisfies (K$\dot{-}$1)–(K$\dot{-}$4) and (K$\dot{-}$6). The agent's beliefs are closed under logical consequence in in a logic W which has a single inference rule (generalised modus ponens):

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

where δ is a substitution function which replaces all free variables of a formula with constants.

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].

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 distinguish *independent* beliefs, beliefs which have at least one non-inferential justification (i.e., a justification with an empty support), such as beliefs acquired by perception and the literals in the belief base when the agent starts. 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 $0, \ldots, m$, 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:

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

where $j_0, \ldots, j_n$ are all the justifications for A, and define the quality of an inferential justification to be that of the least preferred belief in its support: [8]

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

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' [25] 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 dependencies graph of A.

We define the *worth* of a set of literals Γ 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 Γ, then for any other set of literals Γ' such that $K - \Gamma'$ does not imply A, $worth(\Gamma) \leq worth(\Gamma')$.*

[8] Literals with no supports (as opposed to an empty support) are viewed as having an empty support of the lowest quality.

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 $kr + n$.

5.2 Revision

In the previous sections we described how to contract by a belief. Now let us consider revision, which is adding a new belief in a manner which does not result in an inconsistent set of beliefs.

If the agent is a reasoner in classical logic, revision is definable in terms of contraction and vice versa using Levi identity $K \dot{+} \overset{df}{=} (K \dot{-} \neg A) + A$ and Harper identity $K \dot{-} A \overset{df}{=} (K \dot{+} \neg A) \cap K$ (see [13]).

However, revision and contraction are not inter-definable in this way for an agent which is not a classical reasoner, in particular, a reasoner in a logic for which it does not hold that $K + A$ is consistent if, and only if, $K \not\vdash \neg A$. If we apply the Levi identity to the contraction operation defined earlier, we will get a revision operation which does not satisfy the belief revision postulates. One of the reasons for this is that contracting the agent's belief set by $\neg A$ does not make this set consistent with A, so $(K \dot{-} \neg A) + A$ may be inconsistent.

Instead, we define revision of the set of literals in the belief base K by A as $(K + A) \dot{-} \bot$ (add A, close under consequence, and eliminate all contradictions).

```
Algorithm: revision by A

Add A to K;
apply all matching plans;

while there is a pair (B, ~B) in K:
   contract by the least preferred member of the pair
```

In [2], it is shown that this definition of revision satisfies all of the basic AGM postulates for revision below apart from (K$\dot{+}$2):

(K$\dot{+}$1) $K \dot{+} A = Cn(K \dot{+} A)$
(K$\dot{+}$2) $A \in K \dot{+} A$
(K$\dot{+}$3) $K \dot{+} A \subseteq K + A$
(K$\dot{+}$4) If $\{A\} \cup K$ is consistent, then $K + A = K \dot{+} A$[9]
(K$\dot{+}$5) $K \dot{+} A$ is inconsistent if, and only if, A is inconsistent.
(K$\dot{+}$6) If $Cn(A) = Cn(B)$, then $K \dot{+} A = K \dot{+} B$

[9] We replaced '$\neg A \notin K$' with '$\{A\} \cup K$ is consistent' here, since the two formulations are classically equivalent.

6 Belief Revision in *Jason*

Future releases of ***Jason*** will include an alternative definition of the brf() method discussed in Section 3.2 which implements the belief revision algorithm presented above. A belief to be added to the belief base, passed to this new implementation of brf, may be discarded or may result in the deletion of some other belief(s) in order to allow the new belief to be consistently added to the belief base. Which beliefs are effectively deleted is determined by a user-specified preference order (see below).

The only change to the AgentSpeak interpreter code that was necessary to facilitate the implementation of the belief revision algorithm, was to explicitly include in any internal belief change, the label of the plan that executed the belief change. For example, if at a particular reasoning cycle, the intended means (i.e., plan instance) chosen for execution is "`@p1` te `:` ct `<- +b.`", the belief b is annotated with "`plan(p1)`" (in addition to `source(self)`, as normally) before adding b to the belief base.

The graph used by the belief revision algorithm is implemented in terms of two lists for each belief: the "dependencies list" (the literals that allowed the derivation of the belief literal in question), and the "justifies list" (which other beliefs the literal in question justifies, i.e., it appears in their dependencies list).[10] Each belief to be added has an annotation "`plan()`" recording the label of the plan instance that generated it, which can be used to retrieve the necessary information regarding the antecedents of the belief from the plan library (together with the unifier used in that plan's instance in the set of intentions). For example, if the plan that generated the belief change, say $+bl$, has the form "`@p` te `:` l_1 `& ...&` l_n `<-` bd", where te is a triggering event and bd a plan body, the support list of the justification is simply the (ground) literals from the plan context, "[l_1,...,l_n]". Note that 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. Further, for each literal in $l_1, \ldots, l_n$ 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 [23]), and in case of information from similar sources, it gives preference to newer information over older information (this is why the time when a justification was inserted is also annotated, as explained above).

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, e.g., plans 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 consis-

[10] Note that the "dependencies" and "justifies" lists are associated with each unique belief, i.e., a ground belief atom and the annotations with which it is asserted into the belief base, rather than the internal ***Jason*** representation of the belief which holds all annotations for a given ground belief atom in a single list.

tent 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 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.

7 An Example

To illustrate the importance of belief revision in the context of AgentSpeak, we present a simple example 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 $\langle \texttt{ag1}, \mathit{tell}, \texttt{salesUp}(\texttt{c1}) \rangle$ 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 tend to lead to 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 available contraction preference relation, it is not difficult to see that in this instance, the algorithm would contract `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)` $\wedge$ `~goodToBuy(c1)`) 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 require major programming efforts from the user. 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, with the user choosing the *coherence style* of the algorithm, removing `wellManaged` would *not* remove `goodToBuy`. Although in this example the reason-maintenance style is clearly more adequate, in other applications the coherence style might be more useful. In any case, it is clear that without the use of an automatic belief revision algorithm, it would be very difficult for a programmer to ensure such kind of revision would occur appropriately at all times. This would require that the programmer developed an application-specific brf method, or else writing specific plans to handle all possible events (due to belief changes) that might affect any such inferences.

8 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 dramatically increase the need for automatic belief revision. We motived the choice of a polynomial-time belief revision algorithm and described its integration into the ***Jason*** AgentSpeak interpreter. We also gave a simple example which illustrates the utility of such an automatic belief revision mechanism in a practical multi-agent system application, 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.

We are aware of a number of limitations of the work presented here. In future work, we plan to further explore the issue of the interaction of belief revision with the ***Jason*** extension that allows the belief base to refer to OWL ontologies and uses ontological reasoning as part of the AgentSpeak interpreter [17], and to address the issue the inferences that occur from complex test goals. 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)*, Utrecht, the Netherlands, July 2005.
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. *IJCIS*, 2006. To appear.
11. J. de Bruijn, A. Polleres, and D. Fensel. Owl lite^{-}. working draft, wsml delieverable d20 v0.1, WSML, 18th July 2004. `http://www.wsmo.org/2004/d20/v0.1/20040629/`.
12. 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.
13. P. Gärdenfors. *Knowledge in Flux: Modelling the Dynamics of Epistemic States*. The MIT Press, Cambridge, Mass., 1988.
14. 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.
15. D. Makinson. How to give it up: A survey of some formal aspects of the logic of theory change. *Synthese*, 62:347–363, 1985.

16. 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.
17. 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)*, Utrecht, the Netherlands, July 2005.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.
26. M. Wooldridge. *Reasoning about Rational Agents*. The MIT Press, Cambridge, MA, 2000.