ASTREA: Answer Sets for a Trusted Reasoning Environment for Agents

Richard Watson[1] and Marina De Vos[2]

[1] Department of Computer Science
Texas Tech University
Lubbock, TX 79409, USA
richard.watson@ttu.edu
[2] Department of Computer Science
University of Bath
Bath BA2 7AY, UK
mdv@cs.bath.ac.uk

Abstract. In recent years, numerous papers have shown the power and flexibility of answer set programming (ASP) in the modeling of intelligent agents. This is not surprising since ASP was developed specifically for non-monotonic reasoning - including common-sense reasoning required in agent modeling. When dealing with multiple agents exchanging information, a common problem is dealing with conflicting information. As with humans, our intelligent agents may trust information from some agents more and than from others. In this paper, we present ASTREA, a methodology and framework for modeling multi-agent systems with trust. Starting from agents written in standard *AnsProlog* , we model the agent's knowledge, beliefs, reasoning capabilities and trust in other agents together with a conflict resolution strategy in CR-Prolog. The system is then able to advise the agent what information to take into account and what to discard.

1 Introduction

The world is full of situations where entities interact with each other and exchange information. Whether the interactions are between humans, machines, or some combination thereof, interaction is often necessary in order for each to achieve their goals. To model such situations it is therefore important to be able to represent and reason about the various agents and their interaction. One of the difficulties in modeling such agents is dealing with conflicting information. Conflicts may arise for a number of reasons. In extreme cases one agent may be intentionally lying to another agent, but it is more often the case that the agent is simply mistaken in its beliefs. An agent, modeling the behavior of a human, should reason based on assumptions just as a human does. Such assumptions may be wrong and, when passed to another agent, may disagree with information held by that agent or received from other agents. Conflicts may even arise

when modeling electrical or mechanical systems. Such systems often rely on sensors to observe the world. If a sensor malfunctions, it may lead to a false belief about the true state of the world.

It is often difficult, if not impossible, to resolve conflicting information in a way that guarantees that the agent's beliefs are correct with respect to what is true in the world. This is a normal state of affairs when considering commonsense reasoning where the primary concern is that the agent's reasoning is rational, even if it turns out later that the agent was wrong. As an example, suppose a person asks two other people to answer a mathematics' problem. One of the persons asked is a first year math student and the other is a professor of mathematics. If the answers are different and the person has no other reason to disbelieve the professor, they should trust the professor over the student. This does not imply the professor was correct, but it is the rational choice under those circumstances.

There are three major topics one must consider when modeling agents: 1) the choice of language, 2) the methodology used for modeling, and 3) the framework under which the agents operate. In this paper we discuss our approach to agent modeling, ASTREA, which stands for "Answer Sets for a Trusted Reasoning Environment for Agents". As stated in the name, for the language we use Answer Set Programming (ASP). For conflict resolution, due to information arriving from different sources, we propose the use of CR-Prolog. Consistency restoring rules are used to indicate whether information supplied by a particular agent should be disbelieved in case of a conflict. Different conflict resolution strategies, based on the trust agents have in each other, can be encoded in CR-Prolog to provide that the most appropriate consistency restoring rules be used.

The rest of the paper is organized as follow. In Section 2 we provide a short introduction to multi-agent systems, answer set programming, and CR-Prolog. In Section 3, we introduce our ASTREA model. After highlighting the motivation behind our methodology, we propose the formal model of an ASTREA agent in Section 3.2. The mapping to CR-Prolog of the agent's knowledge base is detailed in Section 3.2. The different trust-based conflict resolution strategies are discussed in Section 3.3. and comments on future work.

2 Preliminaries

2.1 Multi-Agent Systems

The key contributors to a Multi-Agent System (MAS) are, naturally, agents. Despite the vast number of research papers published in this area, no general consensus exists about a definition of an agent. In this paper we take the definition from [33] which is an adaption of the one given in [34]:

> An *agent* is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objectives.

A MAS contains a number of agents, each with their own objectives and goals, that affect the environment and collaborate to achieve their individual goals. A detailed discussion on MAS and its history can be found in [33]

Conceptually agents operate in an *observe-deliberate-act* loop[10, 7]. In general the loop has the form:

1. observe and integrate observations, gather information (also referred to as beliefs)
2. consider the options available (also referred desires)
3. select goal (also referred to as intentions)
4. plan
5. execute

When implementing these agents one often finds that each phase is implemented by different components which share common knowledge[1, 26].

This paper is concerned with the first step of the loop where the agent makes the observations and reflects upon the beliefs it holds about the state of the world. Among these observations are the things that the agent has been told by other agents. The agent must then decide what can consistently be believed from among those things it was told. This is, of course, not the easiest of tasks. Not only does one have to worry about the agent being told directly conflicting information, but information may also conflict indirectly.

Take, for example, an agent reasoning about cars and getting additional information from two other agents. Suppose the agent asks for information about a particular car and is told by one of his sources that the car has 4 doors and told by the other that the car is a Mazda Miata. While this information is not directly contradictory, if the agent knows that a Maita is a 2-door sports car then it knows that one of the two must be mistaken.

2.2 Answer Set Programming

To model the knowledge and beliefs of the individual agents we have opted for a form of logic programming, called Answer Set Programming (ASP)[23] developed by Michael Gelfond and his colleague Vladimir Lifschitz. Here we only present a short flavor of the language *AnsProlog*, and the interested reader is referred to [8] for in-depth coverage. *AnsProlog* and its extensions have demonstrated[7, 9, 13, 18, 24, 12, 16] that they are a good and useful tool in the domain of multi-agent systems.

AnsProlog is a knowledge representation language that allows the programmer to describe a problem and the requirements on the solutions in an intuitive way, rather than the algorithm to find the solutions to the problem. The basic components of the language are *atoms*; elements that can be assigned a truth value. An atom, a, can be negated using *classical negation* so creating the *literal* $\neg a$. An *inconsistency* occurs when both a and $\neg a$ are true. A literal l can be negated using *negation as failure* so creating the *extended literal*, *not* l. We say that *not* l is true if we cannot find evidence supporting the truth of l. If l is true

then *not* l is false and vice versa. Atoms, literals, and extended literals are used to create rules of the general form: l :– $B, not\ C.$, where l is an literal and B and C are sets of literals. Intuitively, this means *if all elements of* B *are known/true and no element of* C *is known/true, then* l *must be known/true.* We refer to l as the head and $B \cup not\ C$ as the body of the rule. Rules with empty body are called *facts. A* program in *AnsProlog* is a finite set of rules.

The semantics of *AnsProlog* is defined in terms of *answer sets*, i.e. consistent assignments of true and false to all atoms in the program that satisfy the rules in a minimal and consistent fashion. A program has zero or more answer sets, each corresponding to a solution.

When used as a knowledge representation and programming language, *AnsProlog* is enhanced to contain constraints (e.g. $:- b, not\ c.$), cardinality constraints[28] (*e.g.* $n[a_1, \ldots, a_k, not\ b_1, \ldots not\ b_l]m$) and weight constraints $L \leq \{a_1 = w_{a_1}, \ldots, a_k = w_{a_k}, \neg b_1 = w_{b_1}, \ldots, b_l = w_{b_l}\} \leq U$. The first type are rules with an empty head, stating that an answer set cannot meet the conditions given in the body. Cardinality constraints are a short hand notation a non-deterministic choice; for the constraint to hold a number between n and m of literals in the construct need to be contained in an answer set. Weight constraints are similar to cardinality constraints except that each literal is now given a weight. It is the addition of the weight of all the literals that are true which is taking into account. These additions are syntactic sugar and can be removed with linear, modular transformations (see [8]). Variables and predicated rules are also used and are handled, at the theoretical level and in most implementations, by instantiation (referred to as *grounding*).

Most ASP systems are composed of two processes: removing the variables from the program by instantiation with a *grounder*; and computing answer sets of the propositional program with an *answer set solver.* Common grounders are LPARSE [27] and GRINGO [22] while SMODELS[27], CLASP[21] and DLV[20] are frequently used solvers.

2.3 CR-Prolog

Programs do not always return answer sets. While this is exactly the behavior one would expect for certain problems (e.g. if the problem has no solution); in other situations having an answer is essential (e.g. a decision is required). A program fails to produce an answer set when the program is inherently contradictory. Removing or adding selected rules could resolve this contradiction; resulting in the program returning answer sets. While a learning system could learn the rules to add in certain system, it is more than often the designer's responsibility to specify these rules.

CR-Prolog[5, 4] is a knowledge representation language which extends traditional answer set programming with consistency-restoring rules (cr-rules for short). These rules can be added to the program to automatically resolve the contradictions. The use of cr-rules allows for modeling situations that are unlikely, unusual, or avoided when possible. CR-Prolog has been successfully applied in areas like planning and diagnostic reasoning[6].

A CR-Prolog program[3] consists, apart from the normal *AnsProlog* rules, of consistency restoring rules of the form: $r : l \stackrel{+}{-} B, not\ C$ where r is a label for the rule, l is a literal, and both B and C are sets of literals. These rules have a similar meaning as normal *AnsProlog* rules except that they are only added when inconsistencies occur in the standard program. Given the nature of cr-rules, we of course want to add as few[4] of them as possible.

When there are different sets of cr-rules which could be added to resolve an inconsistency – i.e. various answer sets can be obtained – it is possible to add a preference to indicate which rules should be used. This can be done using atoms of the form $prefer(r1, r2)$ where $r1$ and $r2$ are the labels of cr-rules. When this atom is true, it indicates that no solutions using $r2$ should be considered unless no solution with $r1$ can be found. Adding this preference to the program also rules out solutions in which both $r1$ and $r2$ are used.

CRMODELS is the associated answer set solver for CR-Prolog programs. It is constructed on top of LPARSE and SMODELS[27]. The solver starts with checking if the normal part of the cr-program is consistent by calling SMODELS. If inconsistencies occur, CRMODELS iterative adds CR-rules on the basis of the preference relations until answer sets are found.

We will be using CR-Prolog to resolve inconsistencies that occur when beliefs from various agents are merged. Based on the trust an agent has in the other agents, it will be more likely to believe information from one agent than an other.

3 ASTREA

3.1 Motivation

As discussed in the previous section, an agent's knowledge and reasoning capabilities are modeled as a logic program under the Answer Set Semantics. The Answer Set Programming approaches used to model the knowledge of agents have been quite effective. In such approaches the agent's knowledge is modeled as rules in logic programming. The answer sets of the resulting program correspond to the possible sets of beliefs of a rational agent. This works well in a single agent environment, however, when there are multiple agents passing information problems can arise. The problem lies in the absence of any differentiation in how strongly the agent holds each of the beliefs. For illustration, consider the following well-known example. The agent is aware of the existence of a bird, Tweety. The agent also knows that, while there are exceptions, birds can normally fly. If this is the only information the agent has, the agent would rationally believe that Tweety could fly. Suppose the agent is then told by another agent that Tweety cannot fly. This contradicts the agent's beliefs. The question is how to resolve the contradiction. As a human reasoner, it is easy to see that either the agent was missing the knowledge that Tweety was an exception to the rule about

[3] In this paper we will restrict ourselves to non-disjunctive programs (i.e. rules only have one head atom)

[4] With respect to set theoretic inclusion, not cardinality.

flying or the second agent was wrong about Tweety not being able to fly. The contradiction would also be resolved if the agent was wrong about Tweety being a bird. This third possibility is not one that would be normally considered as it may be assumed that Tweety being a bird is not in question. The real question is one of which of the agent's beliefs is the agent sure of, and which are subject to doubt. The formalism presented in this paper will address the agent's confidence in its facts.

Contradiction between what an agent believes and what is it told by others is due to the fact that agents almost always have to reason with incomplete, and sometimes unreliable, information. Reasoning with such information leads to uncertainty. Uncertainty about an agent's knowledge can arise in several different ways. As mentioned before, one possibility is that the agent may be unsure of some of its own "facts". For example, in the situation above, if the statement was "The agent is aware of an animal, named Tweety, which the agent thinks is a bird" then there would be cause to be less than positive about Tweety actually being a bird. It is often the case that agents, human or otherwise, receive their information from sensors which are not infallible. Hence one often hears statements which begin with phrases such as "I thought I just saw...". A second reason for uncertainty is because the agent was using a defeasible rule and has incomplete information. In the situation above, if the agent has no reason to believe the bird is an exception to the rule about flying, the agent believes that Tweety can fly. The agent knows however that it is making an assumption that may be wrong. If on the other hand the agent knew that Tweety was not an exception, then it would be more certain. A third reason that can lead to uncertainty is that the agent may be reasoning using premises that are uncertain. It the agent above was reasoning about a cage for Tweety, they may reason that if a bird can fly then its cage needs to have a roof. As the agent is not certain that the bird can fly, they cannot be certain that the cage needs a roof. Finally, an agent may have uncertainty about a belief because it is something they were told by another agent.

3.2 ASTREA Agents

Formalization An ASTREA framework consists of a number of agents $\mathcal{A}$. Each agent is a 6-tuple of the form $a = \langle id_a, \Pi_a, C_a, I_a, trust_a, \rho \rangle$. Here, the first element, id_a, is a unique name for the agent. We denote the set of all agent names as A_{id}. The second element, Π_a, is the agents knowledge and beliefs written as a standard *AnsProlog* program. The set of literals, C_a, are the facts within Π_a that the agent is certain of. To help establish communication and make it possible to receive information we specify the agents from which agent a can receive information. This is denoted as I_a with $I_a \in 2^{A_{id} \setminus id_a}$. With the possibility of conflicting information, agents also contain a trust relation to specify their confidence in themselves and the agents they communicate with regarding the correctness of the beliefs they hold. The trust function is defined as follows: $trust_a : I_a \cup \{id_a\} \rightarrow \mathbb{N}$. The higher the number the more trusted the agent is. In this paper we assume that agents assign single trust values to communicating

agents but this could easily be extended to a function where trust is also assigned on the basis of topic.

Depending on the type of reasoning that is needed by the agent, the trust relation can be used in a variety of ways to inform the agent's reasoning. In this paper we look at three different trust strategies plus the situation in which the trust relationship is ignored. In the later case, we obtain all possible ways of obtaining a consistent set of beliefs. When the trust relationship is taken into account, we can take the views of the most trusted agent. In case of conflicts at the highest level, we obtain several possibilities and it will be up to the agent to decided upon one. Another option is to give each agent a vote and count the votes for each side of the contradiction and follow the majority. A similar strategy is to use a weighted voted mechanism on the trust level. More details will be given in the section where we model each strategy. Each agent within the ASTREA framework can have its own preference strategy. Its choice is denoted by ρ with $\rho \in \{none, trusted, majority, weighted\text{-}majority\}$. The set of choices can easily be expanded as more trust relations are formalized.

We assume that the information the agent receives from other agents is in response to a request, r, that each request from a given agent has a unique id number, and that such information consists of a set of ground literals. We further assume that the literals in such a response come from a set of common literals. Formally, for an agent a, the set of all responses received by a will be denoted by R_a. Elements of R_a are 3-tuples of the form $r = \langle r_i, a_j, s\rangle$ where r_i is the unique request id that is being responded to, a_j is the name of the agent returning the response, and s is the set of ground literals which make up the response.

In order to resolve inconsistencies, ASTREA agents consider what they have been told by other agents. In doing so the whole set of literals in a response will either be believed or rejected as a group. The reasoning behind rejecting all elements of a response as a group is that, since the set is in response to a request, one can assume that the literals in the set are likely related and therefore if one is rejected the rest are also suspect. Identifying conditions under which one might only partially reject a response are left for future research.

Before discussing the implementation of ASTREA agents, we will first give an example of a possible program, Π, that an agent may have and a response the agent may receive concerning the domain in question. The example is a precisely stated and expanded version of the Tweety bird problem mentioned earlier in the paper. This example will be used throughout the remainder of the paper to illustrate the use of ASTREA.

Example 1 (A traditional method of modeling an agents knowledge in ASP). The agent knows that birds can normally fly, but that there are exceptions. One such exception is penguins. The agent knows that penguins cannot fly. Another exception is wounded birds. Depending on the nature of the injury, a wounded bird may be able to fly or it may not. However, knowing a bird is wounded is enough reason for the agent to withhold judgment whether it can fly or not. The agent has knowledge of two birds, Tweety and Sally. The agent also believes that Sally is wounded. There is a third bird, Sam, that the agent is currently unaware

of. There are other agents who know about Sam, some of whom believe Sam is a penguin.

Using the traditional methods, this agent's knowledge can be represented by the following program Π:

$$
\begin{aligned}
&fly(X) \text{ :- } bird(X), not\ ab(X).\\
&bird(X) \text{ :- } penguin(X).\\
&ab(X) \text{ :- } penguin(X).\\
&\neg fly(X) \text{ :- } penguin(X).\\
&ab(X) \text{ :- } wounded(X).\\
&bird(tweety).\\
&bird(sally).\\
&wounded(sally).
\end{aligned}
$$

Note that another standard method of representing the first rule above is:

$$fly(X) \text{ :- } bird(X), not\ ab(X), not\ \neg fly(X).$$

In this case the third rule, which states that penguins are abnormal, is not needed. Writing the first rule in this way is useful in the single agent case. If the agent knows a bird cannot fly, but does not have any knowledge about the bird being abnormal, the resulting program would be still be consistent with the alternate rule, but not with the one originally given. In a multi-agent situation however, the encoding we gave is preferable. With the information given in the story above, the agent would believe Tweety can fly. If told by another agent that Tweety could not fly, the addition would be consistent with the alternate rule. This is not the behavior we want - the agent should not automatically believe such an assertion without the other agent giving a reason to believe the bird was abnormal.

Given the story, a possible request the agent may make is to ask other agents for the information they have about birds. An example of a possibly response is

$$\langle r_i, a_j, \{\neg fly(tweety), \neg fly(sam), bird(sam), penguin(sam)\}\rangle$$

with r_i the request id and a_j the name of the agent that responded. In this case, the information about Sam is new. It does not contradict anything the agent currently believes so the agent has no reason to disbelieve it. However, the information about Tweety contradicts the agent's belief that Tweety can fly. It is contradictions such as this that ASTREA is meant to resolve.

Modeling Agent's Beliefs In the example above the domain was modeled in a standard way, as is usual in a single agent environment. In this work, however, we need to be able to resolve conflicts between multiple agents. There are several ways we could approach this problem. We could create either: a new semantics; a new methodology for modeling the domain; or a translation from models created

using the traditional approach into new models which meets our needs. In this paper we opt for the third solution and present a translation. This will allow users to program in a formalism they are familiar with and to use existing off-the-shelf tools.

Before we present the translation we will start by highlighting the assumptions we make. First, for this paper we assume a single time instant and only the rules of the agent which concern the agents knowledge. That is, we do not consider rules regarding the actions of the agent. This assumption is not critical, however it does simplify the discussion in the paper. We also assume that the predicate *disbelieved* of arity 2 does not to exist in Π. If such a predicate did exist we could simply use a different, unused predicate in the translation. Again, this assumption just simplifies the discussion. Furthermore, we assume the agents themselves are consistent. In other words, the program representing the agent's knowledge using traditional methods has at least one answer set. This assumption is quite natural since we assume rational agents and it is irrational for an agent to have inconsistent beliefs.

With respect to the translation, we assume that, for the sake of grounding, the set of all constants used by the agents are known to each agent. This will simplify our discussion of the translation. In actual use one would need to modify the translated program if new constants were discovered in communications with other agents. Due to space consideration, rules which are not changed by the translation will be left ungrounded.

Definition 1 (Translation Π_a^T).
Let $a = \langle id_a, \Pi_a, C_a, I_a, trust_a, \rho \rangle$ be an agent. Using Π_a (grounded) and C_a we create the translated program Π_a^T as follows:

For each rule with an empty body (fact) in Π

$$l.$$

we add one the following to Π_a^T:
if $l \in C_a$ then add

$$l. \tag{1}$$

otherwise, two rules are added

$$l \text{ :- } not\ disbelieved(x, id_a). \tag{2}$$
$$r(x, id_a) : disbelieved(x, id_a)\ +\!-\ . \tag{3}$$

where x is a unique identification number for each new rule.
For each rule in Π with a non-empty body

$$l_0 \text{ :- } l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n.$$

(including rules with empty heads) add the rule to Π_a^T *and for each* $m{+}1 \leq i \leq n$*, add the following two rules to* Π_a^T*:*

$$l_i \text{ :- } disbelieved(x, id_a). \tag{4}$$

$$r(x, id_a) : disbelieved(x, id_a) \text{ +- } . \tag{5}$$

where, as before, x *is a unique identification number for each new rule.*

In addition to translating the agents own knowledge, the responses received from other agents must also be translated. The translation of responses, R_a^T, is formed as follows:

Definition 2 (Translation R_a^T).
Let a *be an agent and* R_a *be the set of responses received by that agent from its requests. For each* $r = \langle r_i, a_j, s \rangle \in R_a$ *with* $s = \{l_1, \ldots, l_n\}$ *the rule*

$$r(r_i, a_j) : disbelieved(r_i, a_j) \text{ +- } . \tag{6}$$

is added to R_a^T *and for each*
$1 \leq k \leq n$ *the following rule is also added:*

$$l_k \text{ :- } not\ disbelieved(r_i, a_j). \tag{7}$$

As in the case of the non-translated program, answer sets of the translated program correspond to possible sets of beliefs of the agent. It is possible that information from other agents in response to requests leads to conflicts. In this case CR-Prolog will automatically try to restore consistency by firing cr-rules.

Each cr-rule fired will add a *disbelieved* predicate to the answer set. If a *disbelieved* predicate added is one from a rule of type 2 then it indicates the agent chose to disbelieve an uncertain fact in the agent's original information. If it is from a rule of type 4, it indicates the agent chose to believe a literal occurring under default negation may be true, hence blocking that default rule. Finally, if the predicate is of the form $disbelieved(r, a)$, where r is a request number and a is the name of another agent, then the agent chose to disbelieve the entire response from agent a to request r. By firing some subsets of the cr-rules, consistency will be restored. Notice that, since we assume the agent itself is consistent, in the worst case disbelieving all responses from other agents will necessarily result in consistency. There will often be more than one minimal subset of cr-rules by which consistency can be restored. The resulting answer sets are the possible beliefs of the agent. This will be illustrated by the following example.

Example 2 (Translation and restoring consistency). Recall the Tweety bird story from Example 1. The agent's knowledge was represented by the following program, Π:

$$fly(X) \text{ :- } bird(X), not\ ab(X).$$
$$bird(X) \text{ :- } penguin(X).$$

$$
\begin{aligned}
&ab(X) \text{ :– } penguin(X).\\
&\neg fly(X) \text{ :– } penguin(X).\\
&ab(X) \text{ :– } wounded(X).\\
&bird(tweety).\\
&bird(sally).\\
&wounded(sally).
\end{aligned}
$$

For this example, assume that the current agent is named $agent_0$ and the unique ids used in the rules start from 1. Furthermore, assume that $C_0 = \{bird(tweety), bird(sally), wounded(sally)\}$. Using our translation, the resulting translated program, Π_0^T is:

$$
\begin{aligned}
&fly(X) \text{ :– } bird(X), not\ ab(X).\\
&ab(tweety) \text{ :– } disbelieved(1, agent_0).\\
&r(1, agent_0) : disbelieved(1, agent_0).\\
&ab(sally) \text{ :– } disbelieved(2, agent_0).\\
&r(2, agent_0) : disbelieved(2, agent_0).\\
&ab(sam) \text{ :– } disbelieved(3, agent_0).\\
&r(3, agent_0) : disbelieved(3, agent_0).\\
&bird(X) \text{ :– } penguin(X).\\
&ab(X) \text{ :– } penguin(X).\\
&\neg fly(X) \text{ :– } penguin(X).\\
&ab(X) \text{ :– } wounded(X).\\
&bird(tweety).\\
&bird(sally).\\
&wounded(sally).
\end{aligned}
$$

As mentioned, for space rules were kept unground when possible.

If then, in response to a request with id 4, the agent receives information from another agent, $agent_1$, stating that they believe that Tweety cannot fly but Sally can (i.e. $R_0 = \{\langle 4, agent_1, \{\neg fly(tweety), fly(sally)\}\rangle\}$), then R_0^T is:

$$
\begin{aligned}
&\neg fly(tweety) \text{ :– } not\ disbelieved(4, agent_1).\\
&fly(sally) \text{ :– } not\ disbelieved(4, agent_1).\\
&r(4, agent_1) : disbelieved(4, agent_1) \text{ +– } .
\end{aligned}
$$

If not for the cr-rules, the resulting program, $\Pi_0^T \cup R_0^T$ would entail both $fly(tweety)$ and $\neg fly(tweety)$ and hence it would be inconsistent. However, using CR-Prolog, one of the two cr-rules will fire, adding either $disbelieved(1, agent_0)$ or $disbelieved(4, agent_1)$. Firing the first cr-rule and adding $disbelieved(1, agent_0)$ corresponds to the agent deciding that they may be wrong about Tweety not being abnormal with respect to flying. It is then safe for the agent to believe

what they were told by the other agent; that Tweety could not fly and Sally could. If the second cr-rule was fired instead, it would correspond to the case when the agent chooses to disbelieve what they had been told by the other agent. As a result, the agent would go on believing that Tweety could fly and would hold no belief one way or the other as to Sally's ability to fly. Even though the information about Sally does not conflict with the beliefs of the agent, the whole response is rejected. Notice that if a response from a different agent had stated that Sam was a penguin and could not fly, then the agent would believe the response in either case as it would not cause any contradictions.

Without adding any additional code, the answer sets returned will correspond to all possible minimal ways of resolving the contradiction. There is no preference given between them. This is the behavior the agent will have if the agent's choice of preference strategy, ρ, equals *none*.

In general, if $\rho = none$ the agent computes its possible sets of beliefs by using CR-Prolog to find the answer sets of $\Pi_a^T \cup R_a^T$. Each answer set is a possible set of beliefs of the agent. In the subsections that follow the other trust relations will be shown.

3.3 Building on Trust

Recall from Section 3.2 that each agent has its own trust function. Two of the four trust relations given below require that the values of this trust function be encoded. If $a = \langle id_a, \Pi_a, C_a, I_a, trust_a, \rho \rangle$ is an agent where $id_a = agent_0$ and $I_a = \{agent_1, \ldots, agent_n\}$ then we define T_a as the program consisting of all facts of the form $trust(agent_i, t)$. where $0 \leq i \leq n$ and $trust_a(agent_i) = t$. Recall that the agent will quantify the trust it has in its own ability.

Distrust Lowest Trust Levels First The first preference relation presented here is one in which the agent prefers to disbelieve information from agents it trusts less when they are in conflict with information from agents it trusts more. This corresponds to the agent having trust strategy $\rho = trusted$. This strategy is encoded using the following program, denoted by Tr:

$$\begin{aligned} prefer(r(N1, A1), r(N2, A2)) \ :- \ & trust(A1, T1), trust(A2, T2), T1 < T2, \\ & disbelieved(N1, A1), not\ disbelieved(N2, A2). \\ prefer(r(N1, A1), r(N2, A2)) \ :- \ & trust(A1, T1), trust(A2, T2), T1 < T2, \\ & not\ disbelieved(N1, A1), disbelieved(N2, A2). \end{aligned}$$

In order to compute beliefs of an agent using this strategy, CR-Prolog is used on the program $\Pi_a^T \cup R_a^T \cup T_a \cup Tr$.

Example 3 (Trust Strategy "Trusted"). Consider an agent, $agent_0$, Π_0^T as in example 2, and T_0 as follows:

$$\begin{aligned} & trust(agent_0, 2). \\ & trust(agent_1, 3). \\ & trust(agent_2, 1). \end{aligned}$$

Notice this means that the agent trusts $agent_1$ more then they trust themselves, but $agent_2$ least of all. Suppose, in response to a request, the agent had $R_0^T =$

$$\begin{array}{l}
\neg fly(tweety) \text{ :– } not\ disbelieved(4, agent_1). \\
r(4, agent_1) : disbelieved(4, agent_1) \ +\!\!- \ . \\
fly(tweety) \text{ :– } not\ disbelieved(4, agent_2). \\
r(4, agent_2) : disbelieved(4, agent_2) \ +\!\!- \ .
\end{array}$$

In other words, in their responses $agent_1$ said Tweety cannot fly but $agent_2$ says Tweety can.

When the resulting program, $\Pi_0^T \cup R_0^T \cup T_0 \cup Tr$, is run using CR-Prolog there is only one answer set. The answer set contains $\neg fly(tweety)$ and both $disbelieved(1, agent_0)$ and $disbelieved(4, agent_2)$. This is the desired result as $agent_1$ said Tweety could not fly and $agent_1$ is the most trusted agent.

If there is a conflict between two equally trusted agents and there is no agent with a higher trust level that resolves the same conflict then there will be multiple answer sets. Note that, if there are multiple requests, an agent may be believed on some of its responses and disbelieved on others.

Trust the Majority Another trust relation an agent might use is to choose to trust the majority ($\rho = majority$). In this case there is no need to specify specific trust levels. In case of a conflict the agent minimizes the number of responses it needs to disbelieve, hence believing the majority. As it is very difficult to tell which *disbelieved* statements within Π_a^T correspond to a given response, if anything is disbelieved within Π_a^T then the agent counts itself as one response disbelieved. In order to implement this relation, the following program, Mv is used:

$$\begin{array}{rcl}
agentvote & \text{:–} & disbelieved(N, id_a). \\
count(C + V) & \text{:–} & C\{disbelieved(J, K) : request(J) : agent(K)\}C, \\
 & & V\{agentvote\}V. \\
ac(C) : countallowed(C) & +\!\!- & . \\
prefer(ac(C1), ac(C2)) & \text{:–} & C1 < C2. \\
\neg count(C) & \text{:–} & not\ countallowed(C).
\end{array}$$

Notice that here we also need predicates $request(J)$ and $agent(K)$ for each request id, J, and agent name, K, other than id_a. The program works by forcing an inconsistency unless the number of responses disbelieved (plus one if the agent has to disbelieve anything in its own knowledge and beliefs) is not equal to the allowed count. The allowed count is controlled by a cr-rule with a preference for answer sets with lower counts. As a result the agent gets the answer set(s) with the minimum count. As the trust function is not used here, the beliefs are computed by passing $\Pi_0^T \cup R_0^T \cup Mv$ to CR-Prolog.

Example 4 (Trust Strategy "Majority"). Consider an agent, $agent_0$, with Π_0^T as in Example 2, and R_0^T as in example 3. In this case, since both the agent itself and $agent_2$ believe the Tweety can fly and only $agent_1$ believes that Tweety cannot, there is only one answer set. In that answer set Tweety can fly, as that was the majority.

It is interesting to note, however, that if $agent_1$'s response had been that Tweety cannot fly and furthermore Tweety is a penguin, then there would have been two answer sets. One, as before, in which Tweety can fly, and one in which Tweety is a penguin and hence cannot fly. This is due to the fact that $agent_1$'s beliefs were only in conflict with the agents beliefs in the first case because $agent_1$ said that Tweety could not fly but did not give any reason for $agent_0$ to believe Tweety was abnormal. By adding that Tweety was a penguin, $agent_1$'s response is no longer in conflict with the beliefs of $agent_0$. This is because, if $agent_0$ accepts what $agent_1$ replied, it assumes it was simply missing the information that Tweety was a penguin and therefore an exception to the rule about flying.

A Weighted Majority Wins The third trust relation is also based on a majority, however this time, agents with higher trust levels carry more weight than those with lower trust levels when deciding the majority. It can be viewed as a voting system where some agents may get more votes than others. This is the strategy used when $\rho = \textit{weighted-majority}$. For this, the program, Wm is:

$$
\begin{array}{rcl}
\textit{vote}(R, A, V) & :- & \textit{disbelieved}(R, A), \textit{trust}(A, V). \\
\#\textit{weight vote}(J, K, L) & = & L. \\
\textit{agentvote} & :- & \textit{disbelieved}(N, id_a). \\
\#\textit{weight trust}(id_a, T) & = & T. \\
\#\textit{weight agentvote} & = & \textit{trust}(id_a, T). \\
\textit{count}(C + V) & :- & C\{\textit{vote}(J, K, L) : \textit{request}(J) : \\
 & & \textit{agent}(K) : \textit{numvotes}(L)\}C, \\
 & & V\{\textit{agentvote}\}V. \\
\textit{ac}(C) : \textit{countallowed}(C) & +- & . \\
\textit{prefer}(\textit{ac}(C1), \textit{ac}(C2)) & :- & C1 < C2. \\
\neg\textit{count}(C) & :- & \textit{not countallowed}(C).
\end{array}
$$

The code is similar to the code for the previous relation with a few changes. First, and most importantly, the trust level returned by the agent's trust function for each agent is the number of votes that agent receives. Next, the predicate $\textit{numvotes}(L)$ must exist for each value L in the range of the agent's trust function. Finally, weight rules from SMODELS are used to assign weights to each vote. The program works similarly to that in the previous example except here total number of votes is minimized rather than simply the number of responses disbelieved. As we use the trust function in this relation, the CR-Prolog program used is $\Pi_0^T \cup R_0^T \cup T_a \cup Wm$.

Example 5 (Trust Strategy "Weighted-Majority"). Consider an agent, $agent_0$, with Π_0^T as in example 2, and both R_a^T and T_a as in example 3. The agent and $agent_2$ both agree that Tweety can fly. In order to accept this belief, $agent_1$ would have to be disbelieved. Since $agent_1$ has 3 votes, that is 3 votes against this answer. $agent_1$ says Tweety cannot fly. Accepting that response would require that both the agent itself and $agent_2$ be disbelieved. Together they also have 3 votes. There is a tie so there are two answer sets, one in which the agent believes Tweety can fly, one in which the agent believes Tweety cannot.

If, as in the previous example, $agent_1$ had added that Tweety was a penguin then in the case where Tweety cannot fly, the agent is not in conflict so there is only one vote against. Therefore there is only one answer set, the one in which Tweety cannot fly.

4 Related Work

The ASP community has a vast body of research in the area of representing either preferences, order, priority, or trust in answer set programs. Some researchers allow preferences expressed on the level of atoms, within the program [11], or on the Herbrand base [29], while others define them through rules [14, 17]. In some systems the preferences are used to select the most appropriate solution after the models are computed [31], while others use them for computing their solutions [19]. It is beyond the scope of this paper to give a detailed overview of all those systems. In CR-Prolog, preference is based on rules. Because of our transformation of the initial *AnsProlog* program, the rules on which the preferences are defined are facts. So one could argue that due to construction we have obtained an ordering on atoms. Each of these atoms is a *disbelieved* atom indicating that all information received from a certain agent about a certain request is ignored. The preferences used in CR-Prolog are tightly linked with its consistency restoring mechanism. While we would have been able to encode the preference in other formalisms using different transformations, we believe that the CR-Prolog's philosophy, believe everything unless you have no other option, fitted in the most natural way.

One of the ways of dealing with contradiction between information supplied from agents that we presented used a voting mechanism. To our knowledge, [25] is the only other paper to describe voting mechanisms in answer set programming. In that paper voters provide a partial preference relation over the set of candidates and three voting mechanism are provided: Borda, plurality, Condorcet. In our paper, voting only takes place when a contradiction takes place and votes are distributed on either side of the contradiction. It is either a straight majority vote or the preference relation is used to assign weighted votes.

Multi-agent systems is a very active research domain, but most of research focuses on agents' actions. While there is research on modeling agents' beliefs [32], very little research has been taken place so far in updating the beliefs of the agents.

[15] proposes an agent architecture based on intelligent logic agents. Abduction is used to model agent reasoning. The underlying language is LAILA, a logic-based language which allows one to express intra-agent reasoning and inter-agent coordination coordinating logic based agents. No preferences between sources can be expressed. Using CR-Prolog together with our *disbelieved* atoms mimics up to a certain point abductive behavior, as we try to deduce who is or could be responsible for the contradiction in the belief set.

In [30], the authors use program composition to model agent cooperation and information sharing. We believe that is not a viable way in large multi-agent systems due to the size of the program. Furthermore, one could imagine agents being reluctant to share all the information (including possible private information) with others.

In the Minerva architecture [26], the authors build their agents out of sub-agents that work on a common knowledge base written as a MDLP (Multi-Dimensional Logic Programs) which is an extension of Dynamic Logic Programming. Our agents do not work with a common knowledge base; each agent decides what she wants to keep private or make available. Minerva does not restrict itself to modeling the beliefs of agents, but allows for full BDI-agents that can plan towards a certain goal. It would be interesting to see if this can also be incorporated in our system.

5 Conclusions and Future Work

In this paper we proposed a new methodology for modeling and reasoning about multi-agents system in which the agents exchange information that is possibly contradictory. The information processing part of the agents uses answer set programming to describe the current knowledge, beliefs and capabilities. When new data is added as a result of a series of requests, the program is translated into CR-Prolog program that resolves possible inconsistencies. Our framework currently offers four different strategies to select the most appropriate conflict resolution recommendations. Two of these are based on the trust agent's place in themselves and communicating agents when it comes to supplying correct information.

One area for future work concerns knowledge versus belief. With new and possibly contradicting information coming from other agents and an ever-changing environment, it is important that agents make a distinction between knowledge, pieces of information that cannot be refuted, and beliefs, information that can be changed over time. This was dealt with to some extent in the formalism presented in that the agent can differentiate between facts it is certain of and those it is not. But what about information passed in a response? A responding agent could separate their response into two parts; the part it knows to be true and the part that it only believes. If we assume that if an agent says it "knows" a literal then the literal must be true in the real world, then the receiving agent could always accept the known part of the response even if it rejected the believed part. The next question that arises is "how does an agent tell the difference

between what it knows and what it believes?" Answering these questions and incorporating the answers into ASTREA could result in a more robust system. We have some preliminary work on these questions. Our current approach involves a larger, more complex translation of the agent's knowledge. It is not presented here due to space constraints.

A somewhat related topic is the question of when can an agent accept part of a response while rejecting the rest. When the parts in question are beliefs this is a much more difficult question. In our examples in this paper, we were concerned with birds and their ability to fly. As a human reasoner we can see that a response which contained information about two different birds could be split. The question is how to incorporate that reasoning ability in the agents. One approach may be to limit requests to try to prevent the responses from containing information that was not closely related. That approach, however, seems to be overly restrictive. For us partial acceptance is an open question.

Another open question concerns incorporating knowledge from responses into the agent's facts. At present new information gained from other agents as a result of a request for information is not added to the agent's knowledge base for future use. Neither is the program updated when the agent finds out that some of its beliefs are false. CR-Prolog guarantees that if disbelieved information is removed, a consistent program remains. However, difficulties arise when CR-Prolog returns with several equally preferred ways of restoring consistency. In the future, we plan to look at different update strategies. It is, of course, theoretically possible to continue to store responses and reason with them as a separate component. In human reasoning however, there comes a time in which the agent moves beliefs from "things they were told" into "things they believe" and no longer try to remember where they heard them. Deciding when and how to do this in ASTREA agents is an interesting question for future consideration. In [2], the authors present dynamic logic programs to capture a knowledge base that changes of over time. EvoLP [3] is an extension of this language designed for multi-agent system. While both deal with inconsistencies, neither consider trust to determine the update. It will be interesting to see if both we can combine updates with trust.

At present agents only communicate facts. One could envisage situations in which agents want to communicate rules as well. Using CR-Prolog, it should be easy to incorporate this. As mentioned earlier, preference relations could be expanded to specify preferences not only on the basis of the agent but also on the topic.

In this paper CR-Prolog was only used as a tool. The research did, however, inspire several ideas about potential modifications to CR-Prolog. While the current version of CR-Prolog can use CLASP as its solver, CR-Prolog itself is tightly coupled to LPARSE. As a result, some new constructs available in CLASP, such as aggregates, cannot be used. Such aggregates would have been useful in encoding some of the trust relations presented in this paper. A new version of CR-Prolog, which was either based on GRINGO or which allowed the easy addition of new language feature, is a possible area for future research.

Another direction for further research on CR-Prolog concerns preferences. One of the trust relations presented in this paper was one in which, in case of conflict, you prefer to disbelieve agents with lower trust levels over those with higher trust levels. An obvious way to model this would be to have the rule

$$prefer(r(N1, A1), r(N2, A2)) :- trust(A1, T1), trust(A2, T2), T1 < T2.$$

where $A1$ and $A2$ are agents identifiers, $N1$ and $N2$ are either unique agent rule identifiers or unique request identifiers, and $T1$ and $T2$ are trust levels. Unfortunately, this would not give the desired results. In the current version of CR-Prolog, given two cr-rules, $r1$ and $r2$, if rule $r1$ is preferred to rule $r2$ then if there is a model which uses $r1$ but not $r2$ it is preferred to a model which uses $r2$ but not $r1$. However, it also disallows models which use both $r1$ and $r2$. Suppose there are three agents, $a1, a2$, and $a3$ with $a1$ being the least trusted and $a3$ being most trusted. For the desired trust relation, disbelieving $a1$ is preferred to disbelieving $a2$, but it may be necessary to disbelieve both if they are both in conflict with $a3$. The ability to use different preference relations within CR-Prolog would enhance its abilities and would allow for more elegant problem solutions in some cases.

References

1. Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. The socs computational logic approach to the specification and verification of agent societies. In Corrado Priami and Paola Quaglia, editors, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 314–339. Springer Berlin / Heidelberg, 2005.
2. J. J. Alferes, F. Banti, A. Brogi, and J. A. Leite. Semantics for dynamic logic programming: A principle-based approach. In Vladimir Lifschitz and Ilkka Niemel, editors, *LPNMR'03*, volume 2923 of *Lecture Notes in Computer Science*, pages 8–20. Springer Berlin / Heidelberg, 2003.
3. Jose Alferes, Antonio Brogi, Joao Leite, and Luis Pereira. Logic programming for evolving agents. In *Cooperative Information Agents VII*, volume 2782 of *Lecture Notes in Computer Science*, pages 281–297. Springer Berlin / Heidelberg, 2003.
4. Marcello Balduccini. Answer set based design of highly autonomous, rational agents. Phd thesis, Texas Tech University, December 2005.
5. Marcello Balduccini and Michael Gelfond. Logic programs with consistency-restoring rules. In *AAAI Spring 2003 Symposium*, pages 9–18, 2003.
6. Marcello Balduccini and Michael Gelfond. The aaa architecture: An overview. In *AAAI 2008 Spring Symposium on Architectures for Intelligent Theory-Based Agents*, 2008.
7. Chita Baral and Michael Gelfond. Reasoning agents in dynamic domains. In Jack Minker, editor, *Logic Based Artificial Intelligence*, pages 257–279. Kluwer Acedemic Publishers, 2000.
8. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
9. Chitta Baral and Michael Gelfond. Reasoning agents in dynamic domains. In *Logic-based artificial intelligence*, pages 257–279. Kluwer Academic Publishers, 2000.

10. Michael. E. Bratman. What is intention? In Philip R. Cohen, Jerry L. Morgan, and Martha E. Pollack, editors, *Intentions in Communication*, pages 15–32. MIT Press, 1990.
11. G. Brewka, I Niemelä, and M Truszczynski. Answer set programming. In *International Joint Conference on Artificial Intelligence (JCAI 2003)*. Morgan Kaufmann, 2003.
12. Francesco Buccafurri and Gianluca Caminiti. A social semantics for multi-agent systems. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *LPNMR'05*, volume 3662 of *Lecture Notes in Computer Science*, pages 317–329. Springer, 2005.
13. Francesco Buccafurri and Georg Gottlob. Multiagent compromises, joint fixpoints, and stable models. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, volume 2407 of *LNCS*, pages 561–585. Springer, 2002.
14. Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Disjunctive ordered logic: Semantics and expressiveness. In Anthony G. Cohn, Lenhard K. Schubert, and Stuart C. Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 418–431, Trento, June 1998. Morgan Kaufmann.
15. Anna Ciampolini, Evelina Lamma, Paola Mello, Francesca Toni, and Paolo Torroni. Cooperation and competition in alias: A logic framework for agents that negotiate. *Annals of Mathematics and Artificial Intelligence*, 37:65–91, 2003. 10.1023/A:1020259411066.
16. Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and analysing agent-based social institutions using answer set programming. In *Selected revised papers from the workshops on Agent, Norms and Institutions for Regulated Multi-Agent Systems (ANIREM) and Organizations and Organization Oriented Programming (OOOP) at AAMAS'05*, volume 3913 of *LNCS*, pages 99–113. Springer, 2006.
17. Marina De Vos, Owen Cliffe, Richard Watson, Tom Crick, Julian Padget, Jonathan Needham, and Martin Brain. T-LAIMA: Answer Set Programming for Modelling Agents with Trust. In *European Workshop on Multi-Agent Systems (EUMAS05)*, pages 126–136, December 2005.
18. Marina De Vos and Dirk Vermeir. Extending Answer Sets for Logic Programming Agents. *Annals of Mathematics and Artifical Intelligence*, 42(1–3):103–139, September 2004. Special Issue on Computational Logic in Multi-Agent Systems.
19. J. Delgrande, T. Schaub, and H. Tompits. Logic programs with compiled preferences. In W. Horn, editor, *European Conference on Artficial Intelligence*, pages 392–398. IOS Press, 2000.
20. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, San Francisco, California, 1998.
21. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 386–392. AAAI Press/The MIT Press, 2007. Available at http://www.ijcai.org/papers07/contents.php.
22. M. Gebser, T. Schaub, and S. Thiele. GrinGo: A New Grounder for Answer Set Programming. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *LPNMR*, volume 4483 of *LNCS*, pages 266–271. Springer, 2007.

23. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
24. Michael Gelfond and Ricardo Morales. Encoding conformant planning in a-prolog. In *DRT'04*, 2004.
25. K. Konczak. Voting theory in answer set programming. In M. Fink, H. Tompits, and S. Woltran, editors, *Proceedings of the Twentieth Workshop on Logic Programmin (WLP'06)*, number INFSYS RR-1843-06-02 in Technical Report Series, pages 45–53. Technische Universität Wien, 2006.
26. J. A Leite, J. J. Alferes, and L. M. Pereira. Minerva - a dynamic logic programming agent architecture. In *Intelligent Agents VIII*, number 2002 in LNAI, pages 141–157. Springer, 2002.
27. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.
28. Ilkka Niemel and Patrik Simons. Extending the smodels system with cardinality and weight constraints. In *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer Academic Publishers, 2000.
29. Chiaki Sakama and Katsumi Inoue. Representing Priorities in Logic Programs. In Michael Maher, editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 82–96, Cambridge, September2–6 1996. MIT Press.
30. Chiaki Sakama and Katsumi Inoue. Coordination between Logical Agents. In Jo ao Leite and Paolo Torroni, editors, *Pre=Proceedings of CLIMI V: Computation logic in multi-agent systems*, pages 96–113, Lisbon, Portugal, September29–30 2004.
31. Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *European Conference, JELIA 2002*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 432–443, Cosenza, Italy, September 2002. Springer Verlag.
32. Cees Witteveen and Gerhard Brewka. Skeptical reason maintenance and belief revision. *Artificial Intelligence*, 61(1):1 – 36, 1993.
33. Michael Wooldridge. *An introduction to multiagent systems*. Wiley, 2002. ISBN: 0 47149691X.
34. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(02):115–152, 1995.