Technical Report

Preserving Confidentiality in Multiagent Systems - An Internship Project within the DAAD RISE Program

Daniel Dilger,
Patrick Krümpelmann,
Cornelia Tadros

05/2013

Part of the work on this technical report has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project A5.

Contents

1 Objectives of the Internship Project

RISE (Research Internships in Science and Engineering) is a summer internship program for undergraduate students from the United States, Canada and the UK organized by the DAAD (Deutscher Akademischer Austausch Dienst). Within the project A5 in the Collaborative Research Center SFB 876 [1], we have planned and conducted an internship project in the RISE program that should support our research. Daniel Dilger was the intern and has been supervised by the PhD students Patrick Krümpelmann and Cornelia Tadros. The aim was to model an application scenario for our prototype implementation of a confidentiality preserving multiagent system and to run experiments with that prototype.

The aim of project A5 in general is the investigation and development of advanced techniques for information processing and for protecting the confidentiality of information in the context of multiagent systems. Information is one of the most valuable commodities, it forms the basis for human decisions and actions in various fields of society such as economics, health care or sciences. As such, information is not only shared or published, but also needs protection. In the database security community there is a long history of research that has been developing mechanisms for publishing information as statistics or database views while preserving the privacy of individuals or the confidentiality of other sensitive information in a database [2]. Apart from the idealized conception in these classical database systems that information is complete and accurate, in many applications the available information is rather incomplete and vague. In such situations information needs both, a non-classical representation and further processing based on expertise for the purpose of decision-making or planning. In the artificial intelligence community there exists a variety of approaches for the representation and processing of incomplete and vague information. Examples are answer set programming (ASP) where both information and expertise are represented by rules, or ordinal conditional functions (OCF) [7] which provide semantics for conditional logical formulas.

The project A5 brings together research on privacy, non-monotonic reasoning and multiagent systems. Its goal is to design agents that are equipped both with a belief component based on ASP or OCF and with mechanisms effectively shielding their sensitive information from other agents. Here, an agent is an autonomous computing system that is capable of processing information (represented in the belief component) in a rational way and of automatically deciding and planning its actions driven by internal goals and based on the belief component.

The specific tasks and objectives of the internship project within the project A5 were as follows.

- The intern should become acquainted with the Java-based agent system Angerona already implemented in our project and the theory of confidentiality-preserving agent interactions.
- The intern should develop and model an agent's beliefs in a complex scenario. As a starting point a model of a complex and interesting scenario already existed but had to be adopted to the research questions within the project A5. The model

describes a criminal story based on [3] where agent roles are investigators, witnesses and criminals.

- The intern should identify a scene from the criminal story with at least one agent having confidentiality interests. The decision making and actions of all agents in the chosen scene should be modeled by adequate desires, intentions and Skills for each agent. Moreover, the agents should be equipped with appropriate speech acts for their communication in the scene.
- The intern should informally describe the confidentiality interests of one agent in this scene. Based on this, the intern should specify confidential pieces of information in an adequate policy language and give semantics to this policy (that is, what the policy should enforce).
- The intern should instantiate and evaluate the modeled scene under the mentioned aspects in the Angerona system, especially, with respect to the confidentiality in the previous task. Appropriate answer set programs for the reasoning components of all agents had to be developed and applied.

2 Background on Project A5

Project A5 aims at developing theories of confidentiality for multiagent systems whereby a defending agent maintains a view of a potentially attacking agent. Hereby knowledge based or epistemic agents are considered, i. e., equipped with symbolic knowledge representation formalisms and in particular with non-monotonic ones with advanced inference and change operators. Epistemic agents with complex inference operators are rarely used in current implementations of multiagent systems. Moreover, models of confidentiality to adequately express the confidentiality interests of an epistemic agent in an open environment are lacking.

In the following, we survey aspects investigated in project A5 that were relevant in the context of the RISE internship.

We consider secrecy from the point of view of an autonomous epistemic agent with incomplete and uncertain information which is situated in a multiagent system [8]. Agents reason under uncertainty about the state of the environment, the reasoning of other agents and possible courses of action. They pursue their goals by performing actions in the environment including the communication with other agents. On the one hand, the exchange of information with other agents is often essential for an agent in order to achieve its goals; especially if the agent is part of a coalition. On the other hand, the agent is interested, or obliged, not to reveal certain information, its secrets. Restriction of communication leads to a loss of performance and utility of the individual agent, coalitions and the whole multiagent system. A good solution of the implied conflict between the agent's goal to preserve secrecy and its other goals is one that restricts communication as little as necessary in order to preserve secrecy. Secrecy of information and in particular the inference problem depend on the representation of information and the appropriate modeling of background information and of the reasoning capabilities of the agents. We show that these can be formalized as properties of the belief change, the attacker modeling and the action selection components of the agent. The intuitive formulation of our notion of secrecy preservation can be formulated as follows: *An agent $\mathcal{D}$ preserves secrecy if none of its secrets Φ that it wants to hide from agent $\mathcal{A}$ is, from $\mathcal{D}$'s perspective, believed by $\mathcal{A}$ after any of $\mathcal{D}$'s actions (given that $\mathcal{A}$ does not believe Φ already).*

We designed a secrecy-preserving agent, i. e., an agent that does not perform a secrecy violating action. Our agent model is based on the well-established belief, desires, intentions (BDI) model. The BDI model [8, 6] has become a leading paradigm in the design of intelligent agents. This model distinguishes between *beliefs*, *desires*, and *intentions* as the main components of an agent's mind, the interactions of which determine its behavior. Roughly, *beliefs* comprise (plausible) knowledge the agent has concerning the current situation and how the world works in general, *desires* encode what the agent wishes to achieve and hence represent possible goals, whereas *intentions* focus on the next actions the agent should undertake to achieve the current goal. The role of *beliefs* in this scenario is to provide the agent with useful information to evaluate the current situation and to find reasonable and effective ways to achieve its goals. In the use of the BDI model in project A5 we strongly focus on the epistemic state and it inference and change operators. That is, the agent's epistemic state contains a representation of its current desires and intentions which guide its behavior.

The complete agent model is illustrated in Figure 1. The agent gets perceptions from other agents in its environment which lead to changes of its beliefs. The agent's beliefs are comprised of its view on the world, its view on other agents and its secrets. Belief operators are used to determine the belief sets BS, i. e., the set of inferences from the view. Different belief operators can be used which represent, e. g., a credulous or a skeptical reasoner. Based on the changed beliefs, the agent generates its current options, or desires, with the generate-options operator. Afterwards, the agent commits to some of its current desires which then form its high-level intentions. The high-level intentions are then broken down into sub-goals by the Subgoal-Generation Operator. In the latter process, the potential violation of secrecy of the sub-goals is evaluated by simulating the implied changes to the beliefs of the agent and in particular on the view of other agents. If some other agent would be able to infer some secret information based on the resulting view of that agent, the sub-goal under consideration is determined to be secrecy violating and alternatives are looked for. The inference of the other agent can be postulated to apply a credulous or skeptical belief operator. This process ends when it has determined a non-secrecy violating atomic intention, i. e., an atomic intention can be immediately achieved by executing an action. The corresponding action is then executed in the environment and the beliefs are changed accordingly.

The project makes use of Answer Set Programming (ASP) and ordinal conditional functions (OCFs) as two candidates for non-monotonic knowledge representation. For the internship ASP was used. ASP allows for intuitive knowledge representation and comes along with several powerful and fast solvers which have proven to be practically usable which makes ASP especially interesting for resource-constrained data analysis. For a short introduction of ASP see Appendix A.

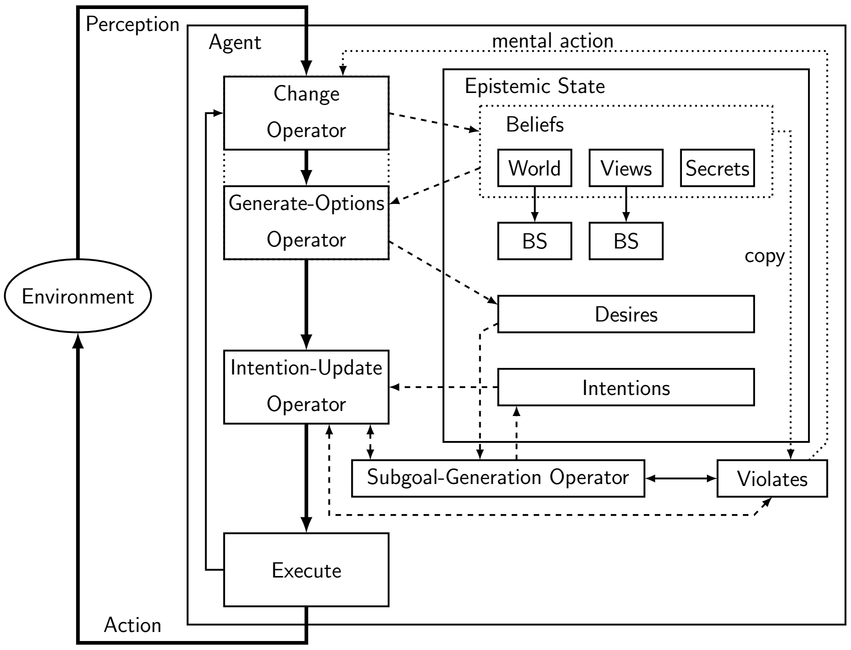

Figure 1: Agent model

We implemented a multiagent system framework called *Angerona* which is based on a versatile plugin architecture. Agents within this framework are epistemic BDI agents based on [5] and its BDI extension whereby both, the knowledge representation and the concrete agent cycle, are flexible. The knowledge representation is based on the interfaces from the *Tweety*[1] library and thereby allows for the use of a variety of formalisms. The plugin architecture allows us to easily define and compare different types of agents and evaluate their performance. The ASP library has been greatly extended in the *Tweety* library and used to implement an ASP plug-in for *Angerona*. An extended BDI cycle for secrecy preservation based on [5] and its extensions have been implemented as well as an ASP plug-in realizing different belief and change operators.

[1]`http://tweety.sourceforge.net/`

3 Internship Report of Daniel Dilger

This document describes the work I performed between May 14th, 2012 and August 3rd, 2012 for my internship project, "Preserving Confidentiality in Multiagent Systems".

My objective in this internship was to develop and model a scenario in which multiple agents are present and at least one secret is at stake for one agent. I was then to implement the scenario using the multiagent simulation framework Angerona, a Java and ASP based framework developed by the department. I was to try to model the behavior of my scenario's agents using existing confidentiality-preserving mechanisms in Angerona, and where necessary extend Angerona with confidentiality-preserving mechanisms from the background theory.

The first section "Description of Modeled Scenario" introduces all information relevant to my scenario. It describes the scenario and the source that inspired it, describes the scenario as it is modeled in Angerona, explains the confidentiality policy of the agent of focus ("Mary") for the scenario, and lists some of the issues that the scenario raises. Formalized definitions important to understanding the document are given in the section that follows.

The section "Observations and Findings" explains some mechanisms which I discovered were necessary for my agents to have so that they would behave as expected for my scenario.

The section "Overview of Changes Made" provides a description of the changes I made to Angerona so that my scenario could be modeled.

The section "Open Issues" describes some shortfalls of Angerona that I discovered while testing my extensions to the framework.

The "Outlook" section gives some suggestions for how to build on the results of my internship.

Finally, test cases for my scenario are provided in Appendix B and pseudocode describing the changes made to Angerona are provided in Appendix C.

Summary of Key Activities

- I developed a scenario to model in a multiagent system under the BDI architecture.
- I modeled this scenario in the Java-based multiagent system *Angerona*.
- In creating the Angerona model I found it necessary to extend Angerona's capabilities.
- The implementation of my scenario highlighted further useful extensions for Angerona.

  In addition to this work for the department, I also presented my progress as of my eighth week at a DAAD RISE conference in Heidelberg (July 7th 2012).

4 Description of Modeled Scenario

This scene comes from Chapter 11 in Agatha Christie's *The Mysterious Affair at Styles*.

Agents involved: Coroner, Mary, maid Dorcas. The Coroner asks questions to Mary, who responds. The maid's actions are assumed to have already occurred.

4.1 The Scene as it is in the Book

At a hearing the courtroom coroner first asks Dorcas what she overheard of the quarrel between Emily Inglethorp and another person, who is generally assumed to be her husband Alfred. When questioned by the Coroner Dorcas reveals that she heard something about a "scandal between husband and wife".

Mary is questioned by the Coroner next. Her questions are the main focus of interest for this scene. She knows that it was her husband John who was arguing with Emily. Thus she has two conflicting goals: to keep her husband's identity secret, and to avoid lying in court.

She decides only to reveal that she did hear the quarrel and that she heard the same fragment that Dorcas revealed. Otherwise she claims she doesn't know anything about the argument (though this is her action in the book, it is worth noting claiming ignorance is a different act from declining to answer, and the former can in fact be considered a lie).

4.2 Angerona Simulation

The scenario represented in Angerona starts in the middle of the book scenario, after the maid has already revealed some information about the argument. The Coroner asks Mary a few open questions, instructing her to tell him everything she overheard during the argument. Mary wants to keep secret that her husband John argued with Emily. She knows three things which were stated by Emily during the argument:

1. "You have lied to me"
2. "You owe everything to me"
3. "A scandal between husband and wife"

She also knows that #3 (and only #3) was already revealed to the Coroner by the maid. The Coroner's questions are represented by the query said(X), to which Mary must either reply with one of her facts like said(youLied) or she must tell the lie that she doesn't know anything. She wants to choose the answer which is least likely to reveal her secret, but she is also capable of saying she does not know anything (or anything else) by responding with dontKnow(said). The Coroner expects to hear at least one piece of information regarding the argument. If he wants to hear more he will ask again.

Mary reasons as follows:

- With right information, Coroner could suspect Alfred, John, or Lawrence
- Alfred and John are married
- All three are financial dependents of Emily
- “A scandal between husband and wife” implies a married man
- “You owe everything to me” implies a financial dependent
- “You have lied to me” is generic and doesn’t single out anyone

For simplicity in implementation the ASP representation of Mary’s reasoning only includes the rules of which statement suggest who, not her underlying reasoning like “John and Alfred are married”. In ASP Mary’s knowledge looks as follows.

```
world {
        said(youLied).
        said(youOwe).
        said(husbandWifeScandal).
        argued(john).
}
view->Coroner {
        said(husbandWifeScandal).
        argued(john) :- said(husbandWifeScandal), not argued(alfred).
        argued(alfred) :- said(husbandWifeScandal), not argued(john).
        argued(john) :- said(youOwe), not argued(alfred), not argued(lawrer
        argued(alfred) :- said(youOwe), not argued(john), not argued(lawrer
        argued(lawrence) :- said(youOwe), not argued(john), not argued(alfr

}
```

Keep in mind that because Mary’s decision-making process is the central concern of the scene, it is sufficient to model her views of the Coroner’s reasoning process without actually giving the Coroner reasoning capabilities. All that the Coroner needs to do is ask the questions which Mary must decide how to answer.

4.3 Mary’s confidentiality policy

Mary’s secret is that her husband John argued. The only speech acts she is capable of is giving answers to an open query about what was said, or to express ignorance about what (or what else, if she has already answered something) was said. She knows three facts regarding what was said, each of which affects the safety of her secret to a different degree.

Her confidentiality policy is shaped by two desires: protecting her secret and avoiding being accused of lying in court.

She wants to keep her secret as strong as possible. That is, of all possible considerations which the Coroner makes, the Mary's secret appears in the smallest proportion of them possible. This **quantification of secrecy strength comes naturally through the use of answer sets**, as each answer set created by her view of the Coroner can model a different consideration he makes. The smaller the percentage of answer sets containing the secret, the better. It's worth noting that this aspect of the confidentiality policy is the only one explicitly represented in the model; all other aspects are implied through Mary's actions or lack thereof.

Importantly, she seeks to minimize the degree with which she weakens the strength of her secret. She does not keep a minimum threshold over which her secret must stay.

If some other person reveals information which weakens her secret's strength, she will adjust her definition of the secret's strength. **Repeating a piece of secrecy-relevant information already revealed by someone else has zero cost to her secret**. She can give up secrecy-relevant information revealed by someone else.

A secondary way in which she protects her secret is by avoiding suspicious behavior as much as possible. "Suspicious behavior" includes any behavior outside of providing an answer when questioned in court. Thus **refusing to answer a question is not an option** under her confidentiality policy, though answering by claiming ignorance is. She also considers repeating herself suspicious, unless it is to reaffirm that she doesn't know anything. **Self-repeating is assigned an infinite cost** and so is never undertaken.

In addition to choosing a safe truthful answer, **Mary's policy also allows for lying**. She doesn't produce lies which provide new information to the Coroner, such as saying "Alfred argued". Instead she only lies by answering "I don't know" to a question. As she does not know what the Coroner could later discover she does not desire to give any information which could later expose her as a liar.

Her desire to avoid being accused of lying explains why she consider revealing secrecy-relevant information at all instead of always answering "I don't know". **She considers it a risk to tell even the lie "I don't know" and assigns a cost to that risk** according to her estimated probability of being exposed a liar later. Unless the cost of revealing secret information exceeds her estimated cost of lying, she will tell the truth. If the cost of lying and telling the truth are the same, she will still prefer the truth.

4.4 Issues raised by the Scenario

Weakening secrets

In this scenario the secret is never revealed in all possible answer sets by any combination of revealed information. The secret is at most present in half of the answer sets. However not all revealed information is equal. Every piece of information allows the secret to exist in a different proportion of answer sets – either in 0, in 1 out of 3, or in 1 out of 2. Thus the secret is not as strong given different pieces of information revealed. The information does not reveal but instead weakens secrecy.

Before secrets can be weakened a quantification of their strength must be given. The strength of a secret is determined by the belief operator assigned to it.

Comparing secrecy weakening to cost of lying

While Mary's desire to keep her husband's role in the argument may suggest that she should always tell this lie, her conflicting desire not to be later found guilty of perjury in court motivates her to tell the truth. She resolves this conflict of interest by weighing the cost of lying to the cost of revealing secrecy-relevant information. To assign a cost to telling a lie, she has to have an estimate of the likelihood that her lie would be revealed. She then has to be able to compare this weighted estimate to the cost of weakening secrecy.

For my implementation the estimated cost of telling a lie is fixed for any lie at any time. It is a constant chosen on the same 0 to 1 scale as the d value representing the strength of a secret.

The cost of affecting many secrets

Some function must exist which converts the level of secrecy weakening to a value which is comparable to Mary's estimated cost of telling a lie. When the cost of telling a lie is expressed on the same scale as secrecy strength is quantified, as in my scenario's implementation, such a function is straightforward for single secrets. The cost of weakening a secret can be expressed as the difference in the d value representing the old and the new strength of the secret. The function must be more sophisticated when multiple secrets are affected by an action. One possibility is to express the cost in terms of the maximum degree of weakening done to a secret. Another possibility, the one currently in Angerona, is to sum up the degrees of weakening. Other scenarios may find other functions more appropriate.

5 Relevant Definitions

5.1 *d* Family of Operators

The term "d Family of Operators" refers to a family of reasoning operators where each member is distinguished by a value d, ranging between 0 and 1. The value d represents the smallest ratio of answer sets containing a fact to total answer sets for the fact to be believed. A d-value of 0 refers to a completely credulous operator and a d-value of 1 refers to a completely skeptical operator. Formally:

d $= 1 - \frac{|\{x: s \in x, x \in a\}|}{|a|}$ where s is the secret in question and a is the set of answer sets.

5.2 Secrecy Weakening

Each secret is assigned a reasoning operator which provides the standard by which the secret is considered known. When the reasoning operator belongs to the d family the strength of the secret can be quantified by its d-value. An operator with a higher d-value corresponds to a higher standard by which a secret is considered known. Specifically, the secret must be present in a lower proportion of answer sets to be considered revealed. The more credulous a reasoning operator allowed without revealing a secret, the stronger the secret is kept. To weaken a secret is to replace the secret's reasoning operator with one of a lower d-value.

5.3 Open and Closed Queries

The terms open and closed queries come from the database community. An open query refers to a question to which the answer is the list of all facts which satisfy the conditions of the question. A closed query refers a question to which a true, false, or unknown value can be answered. In other words, an open query is requesting a variety of facts like "all of the presidents of the United States" whereas a closed query is capable of of only providing true, false, or unknown as an answer to a question like "Was George Washington the first president?"

5.4 Detail Query and Answer Speech Acts

In the courtroom scenario the Coroner asks Mary an open query, "what was said?" However, since the agents in the scenario model humans, and not databases, Mary need not be expected to answer the query with a complete list as a database would. The Coroner instead expects her to do what is more likely for a human and answers only a subset of all facts satisfying the conditions of the question. I refer to this variation of open queries as a "detail query" since only some facts (or "details") regarding the question are expected, rather than all relevant information. For the courtroom scenario, because Mary wants to reveal as little as possible, the subset of possible answers will never have a cardinality greater than one. Formally:

Definition: Detail Query

For every agent A_1, A_2 in a set of agents; `p(x)` *where there exists an* `x` *such that there is a* `p(x)` *in the set of literals, the definition of a detail query is* <A_1 `asks the detail query p` *to* A_2 > *with the meaning:* A_1 *asks for a detailed answer* `x` *to satisfy predicate* `p`

Definition: Detail Answer

For every agent A_1, A_2 in a set of agents; `p(x)` *where there exists an* `x` *such that there is a* `p(x)` *in the set of literals, the definition of a detail answer is* <A_1 `gives the detail answer  s` *for predicate* `p` *to* A_2 > *with the meaning:* A_1 *returns a set* `s` *where* `s` *is a set in which for every element* `x` *there is a corresponding literal* `p(x)` *in the set of literals.*

As an illustrative example of the difference between an open query and a detail query, consider the query "list the presidents of the United States". The expected answer to an open query would be a list containing every president in the history of the US. On the other hand a detail query would only expect a subset of all presidents, as long as at least one president was named.

As a corollarly to the fact that not all answers are expected for one query, it is possible to repeat the query and expect new information.

6 Observations and Findings

In implementing my scenario I found a need for a number of confidentiality-preserving mechanisms present from existing theory.

Contradiction checking

Due to belief revision, enforcement of secrecy for contradictions of current belief base not possible unless a special contradiction-checking mechanism is included.

Secrecy weakening and choice

If secrets can be weakened rather than digitally “revealed” or “not revealed”, it allows choice between secret relevant information.

Secrecy weakening and secrecy dynamism

Secrecy-relevant information given up by secret defender if revealed by another source. This is an automatic result of enabling secrecy weakening.

Evaluation functions of actions in plan

Since Mary wishes to choose an optimal option, among her choices of information to reveal or lying, three new functions had to be introduced to the intention update operator. One function needed to calculate the cost of weakening the secrets affected by an action. It had to determine the cost of not only affecting one secret, but how to combine costs when multiple secrets are affected. Another function needed to calculate the cost of telling a lie. Since the consequences of telling a lie cannot be modeled in my scenario, the function simply assigns an estimated cost to the assumed risk associated with telling a lie. The third function then compares the costs assigned to the agent’s options, and from them chooses the minimal costing one.

Agent histories and self-repeating

The ability to recognize when secrecy-relevant information has been revealed, and to weaken the standard of a secret accordingly, is a desirable feature for agents. A consequence of this ability is a motivation to repeat information which has already been revealed, as once that information has weakened a secret it cannot weaken the secret further. It can be sensible to repeat information in some contexts, such as to reaffirm a statement made by another agent, but self-repeating is usually undesirable. Self-repeating is particularly undesirable in the courtroom scenario, as Mary would appear suspicious if she always repeated herself when asked a question. To prevent self-repeating, agents have to be able to remember what they said previously. If agents keep a history of their actions, they can refer to it to determine whether the action they are currently considering is a repeat.

Some secrecy weakening preempts lesser weakening

Since secrecy strength is based on the proportion of answer sets revealing secret, revealing some information removes the secrecy-relevancy of other information if the answer sets cannot be changed by the new information.

Credulous operators

Any operator other than the purely skeptical needs a policy for contradiction resolution

Semantically different types of strict negation

There is often more than one way in which a statement can be negated, depending on which particular part of the statement is the focus of the negation. For instance, the Coroner may believe that an agent Poirot said that Mary lied. Thestatement "Poirot said Mary lied" would be in his belief set. If that belief were negated, it could be negated as "Poirot did not say she lied", a simple negation of the fact, or as "Poirot said she did not lie" which is a negation by replacing the fact with a contrary one. Only the first type is supported in the simulated scenario.

Utility of realizing own lies

For Mary's courtroom scenario an agent needs to be able to recognize when it is lying so that a cost can be associated with that lie. Having an agent recognize its own lies could also be useful in an extensions of the scenario's contradiction checking mechanism. If an agent wants to distinguish between causing a contradiction to another agent's belief base when being deceptive (by telling a lie) or being honest (by, for example, correcting a mistaken belief held by the other agent), the ability to recognize one's own lies would be useful.

7 Overview of Changes Made

This section is only an overview of the major changes made. For a full description and pseudocode of the algorithms introduced here, please refer to Appendix B. While lengthy, this section does not cover all changes made to the Angerona framework. Some changes were made purely to make the changes listed here function smoothly within the Angerona framework; the unincluded changes can be considered as "overhead" work needed to be performed to extend the capabilities of the framework and not interesting algorithmically. Such changes include modifying the BaseBeliefbase class and adding new supporting classes called "DetailQueryAnswer", "DetailQueryDO", "DetailQueryAnswerDO", and "DetailPerceptionFactory".

7.1 Subgoal Generation

I created a new Subgoal Generation Operator which allows for the asking of multiple questions and the asking and answering of detail queries.

The method "interrogateOtherAgent" produces the questions which an agent may ask .Questions are generated by parsing the names of the agent's desires, which are listed in the simulation file. The names of the desires follow a format "q_textitpredicate(textitarguments)_textiti
(Keep in mind that desires specify specific actions in Angerona; desires are not as abstract or comprehensive as in the theory). The "q_" tells the Subgoal Generation Operator that the desire is a question. The textitpredicate is the question being asked. The parentheses and arguments within are optional. Without them the question will be considered a closed query later in the answering process. To include them there must be only one argument. If the argument is in all caps the argument will be considered a variable, and during the answering process a grounded literal will be found to correspond to that argument, as is done in answer set programming or Prolog (e.g. said(X) could yield said(youLied)). The number at the end of the desire's name is used to enable ordering for questions. Questions are asked in ascending order according to their number. That is, the question with the lowest number will be asked first. Once the questions have been produced, they are packed into subgoals in their specified order. Then, the Intention Update Operator can refer to those subgoals when choosing what the agent will do next.

The method "answerQuery" produces answers for an agent to give when posed with a question. As suggested by the syntax of the desires for questions, the Subgoal Generation Operator also makes distinctions between closed and open query question types. All questions are stated through an instance of the Java class for detail queries, but when formulating an answer the answerQuery method calls a method which refers to the structure of the question. The method "simpleQuery" returns "true" when a close query type is found. It considers any question a closed query when the predicate doesn't have arguments, or when the predicate where all arguments are not stated in all capital letters (e.g.said(youLied)). A question whose predicate has an argument stated in all capital letters (e.g. said(X)) produces a return value of "false" for the simpleQuery method. Within the answerQuery method, a question for which simpleQuery returned true is given true/false answer. Otherwise an answer for every grounded literal matching the question's all caps variable argument is given. For example, said(X) produces the answers said(husbandWifeScandal), said(youOwe), and said(youLied). Lies are also produced according to how the question was categorized. When an answer to a closed query question is produced, a lie which is the logical negation of the truthful answer is also produced. An instance of a class called "Lying Operator" is produced and the truthful answer is passed to a method called "simpleLie". This method returns the answer "false" given "true", and vice-versa. No special lies are produced for open questions. After the information-bearing answers have been produced – a truthful answer and correspond lie for closed query types, one or more truthful answers for open query types – an expression of ignorance is also produced. Once all of these answers have been produced, they are packed into the plan for a subgoal of the agent. When it comes time to choose an action, the agent's Intention Update Operator will refer to these plans to determine the best answer for the agent to give.

7.2 Intention Update Operator

The old Intention Update Operator would iterate through atomic actions and choose the first action whic did not violate confidentiality. I created a new Intention Update Operator which allows multiple options to be considered at once and which chooses an optimal option according to some evaluations specific to my scenario. It contains five methods: the main method processInt, the method isLie, the cost evaluation methods lyingCost and secrecyWeakeningCost, and the minimizing method minimalCosting. The method processInt is the one called by the agent cycle. It produces a list of atomic actions in the agent's subgoals and plans and then calls the other four methods of the class in order to determine which atomic action to perform. If the action is determined a lie by isLie, then a cost according to lyingCost is assigned to the intention (atomic actions belong to the Intention type). Otherwise, a cost is assigned to the atomic action according to the secrecyWeakeningCost method. Once costs have been assigned to all actomic actions in the list, the optimal intention is chosen according to minimalCost. The secrecy weakenings due to the chosen atomic action are sent to the agent's object, and then the intention of the atomic action is returned. The isLie method checks an "honesty" flag stored in the representation of an intention. If the flag is set to false, then the isLie method returns true. Otherwise it returns false. The lyingCost method is passed an intention object. It sets the cost field of that intention to the fixed cost 0.5. A more complicated version of this method would only be possible if some means existed for modeling the consequences of being caught lying. There isn't any such modeling at the moment, and so there is a simple fixed cost. The secrecyWeakeningCost method calculates the cost of an intention which weakens secrets. It gets passed a list of pairs containing a secret affected by the action and the corresponding degree by which the secret has been weakened. The method returns a cost equal to the sum of the degree by which each secret is weakened. Other means to combine the cost of weakening secrets could be considered in the future, such as choosing the maximum degree of weakening rather than summing the degrees or having the degrees of weakening of different secrets have different levels of influence on the cost calculated. Note that by the flow of the processInt method, lies are always excluded from being evaluated by this function. A future Intention Update Operator may want to consider lies that also affect secrets.

The minimalCost method is just a simple minimization function. Given a list of intentions, it chooses the intention with the lowest associated cost.

7.3 Violates Operator

I created a new Violates Operator which is calculates the degree by which secrets are weakened, checks for potential contradictions in an attacker's belief set, and avoids giving the same answer twice. Like most operators in Angerona, my Violates Operator is invoked through a so-called "processInt" method. While the processInt method of the Intention Update Operator returned an intention, the processInt method of violates operators returns a boolean which specifies whether the agent's confidentility policy has been violated (i.e. a secret has been revealed) within the simulation of a specified intention. For my particular violates operator processInt always returns false, as my scenario is concerned with minimizing the degree by which a secret is weakened rather than wheter a secret has become known. The processInt method is useful mainly because it is the means by which the method "processIntAndWeaken" is called. This method calculates the degree by which secrets are weakened and returns a list of SecrecyStrengthPairs, which associate a degree of weakening to a secret. The list of SecrecyStrengthPairs returned is then stored in a field called "weakening" to be accessed later by the intention update operator. Certainly a further extension would be to both calculate the weakenings of an action and to determine based on the secret's reasoning operator whether confidentiality has been violated.

The processIntAndWeaken method first checks if the intended action is an answer, as it only considers answers as able to affect secrets. Then it refers to the agent's history to see if the answer is a repeat of any previous answer it has given. If it is, it considers all answers infinitely weakened. In the future it should probably be possible to determine if the answer is a repeat of the same question, and if there are reasons for self-repeating, such as presenting the same information to a new audience. If the answer passes the self-repeat test, the answer is added to a copy of the logic program of the agent's view of its answer's receiver (e.g. Mary creates a copy of her view of the Coroner to put the new information in). The answer sets corresponding to the simulated logic program are updated accordingly. A check is then made to see if any answer sets exist for the logic program after its expansion by the new information. If there are no answer sets, then a contradiction must have occurred and the weakening violates operator assumes that something bad has happened in its simulation of providing an answer. Therefore it considers all secrets held by the agent to be infinitely weakened. Note that by adding the information directly to a logic program the process of belief consolidation is bypassed altogether. In the future a contradiction checking mechanism such as described here may not be necessary, as a proper revision operator for the simulation would more appropriately simulate the consequences of producing a consequence in the receiver's beliefs. At the moment contradictions are avoided altogether because if consolidated with the belief revision operator currently implemented, facts are preferred over rules in the event of a contradiction. This means that a rule which could reveal to the revealing of a secret is ignored if the complement of the secret information is believed, which is an innapropriate response to contradictions for my scenario; it is assumed that agents give their internal reasoning processes a higher priority over facts, as facts may come from untrustworthy sources such as an agent holding a secret. For example, the Coroner should not give up a rule leading to the secret just because a witness like Mary told him some contradictory fact – for a full example see "Contradiction checking" in the "Observations and Findings" section. A better way to produce this desired preference for rules would be to represent that preference in a new revision operator rather than to use a contradiction checking mechanism in the violates operator. After contradiction checking occurs, another check is made to see if any secrets are present in any answers sets of the attacking agent. If a secret is found present, another method, calculateSecrecyStrength, is called which calculates the proportion of answer sets in which that secret is present. It returns the textitd value of the secret according to this proportion. The processIntAndWeakening then subtracts the new textitd value from the old texitd value of the secret and produces a SecrecyStrengthPair which matches the affected secret to the difference in textitd values. The SecrecyStrengthPair is added to the list which eventually becomes the return value of the method. After the degree of weakening for a secret is calculated, the secret affect and its corresponding degree of weakening is reported to the user. If it is discovered that no secrets are weakened at all, that fact is reported to the user. Finally, the method returns the list of SecrecyStrengthPairs is produced. If no secrets were affected by the simulated answer then the list is simply empty. If a self-repeat or a contradiction is found then a list describing all secrets affected by an otherwise unreachably large value (such as 1000 in this implementation) is returned. Otherwise the list contains the SecrecyStrengthPairs calculated by the final step in the method's execution, in which calculateSecrecyStrength is called and the definition of textitd is invoked. As stated before the result of the method is stored by the processInt method in a "weakenings" field for later reference by the intention update operator.

7.4 Belief Updates Operator

I created a new belief update operator which, in addition to updating beliefs as before, also updates the textitd values of secrets according to the effects of the action chosen by the agent. The operator receives the values by which to update the strength of secrets from the agent object. As stated in the description of the new intention update operator, agent object stores a list of SecrecyStrengthPairs which is set by the intention update operator when an action is chosen.

7.5 Reasoning Operator

I created a new reasoning operator to produce all answers regarding a question. A method "queryForAllAnswers" is called by the subgoal generation operator and passed a question. The method passes that question to a helper method, "findAllAnswers". The helper method generates a list of literals by finding all literals whose predicate matches the predicate of the question. The queryForAllAnswers method takes the list of literals returned by the helper method and bundles each element into the appropriate type for answers to detail queries, and then returns the new list.

7.6 Agent class

I made two changes to the definition of the agent class to faciliate the changes described previously. The first was the addition of a history of actions which the new violates operator refers to in order to check for self-repeating. The actions history is represented as a simple list of actions. The second change was a field which stores a list of SecrecyStrengthPair objects. The belief update operator refers to the field to determine the secrets weakened and the degrees of weakening caused by the action last chosen by the intention update operator.

7.7 Intention class

I added two new fields to the intention class. One was a field specifying whether the intention was “honest”, or not a lie. The honesty field is set by the subgoal generation operator and referred by the intention update operator. The other field added stores the cost the intention update operator associates with the intention. As stated in the “Open Issues” section, in the future both of these changes should be removed in favor of storing the information in the context associated with the intention, as such an approach would be more appropriate for the intended agent cycle for Angerona.

7.8 Skill class

As the Skill class inherits from the Intention class, it also contains the honesty and cost fields. In addition is contains a weakenings field which stores the degree of weakening calcualted and stored by the violates operator. The weakenings value is retrieved by the violates operator after a simulation of running the skill is performed. It is the weakenings field in the Skill object which is accessed by the intention update operator to determine the cost of weakening secrets and, ultimately, to store in the weakenings field of the agent object once an atomic action (which has the type of a Skill) is chosen by the operator.

8 Open Issues

While the core scenario runs as expected, there was unfortunately not enough time to address a few problems with aspects of its implementation or bugs regarding other added features.

8.1 Weakening Secrecy and Completely Skeptical Operators

Currently if a secret has been assigned a completely skeptical reasoning operator – a reasoning operator with a d-value of 0 – revealing the seceret in an answer set will not lead to the secret to be given up. Instead, the secret will be “weakened” by a degree of zero. An internal distinction between a completely credulous reasoning operator and a lost secret must be made. Some assessment for the cost of losing a secret will have to be added in the decision-making process of the intention update operator as well. Note that because a secret is never completely lost in the courtroom scenario, the scenario runs despite the lack of this important feature.

8.2 Lying about Facts in the Negative

When Mary tries to keep a negated fact secret like “Alfred did not argue” (¬alfred_argued), Angerona crashes. Angerona cannot recognize the negated form of a fact as such; it thinks that the fact is a completely new one which hasn't been assigned a signature. For more please refer to Appendix A3, "Bug Demonstration Cases"

8.3 GUI Output

The following are known instances of faulty GUI output:

- A query about a fact with closed arguments, such as “argued(john)?” will be recognized as a closed query and the answer will be chosen accordingly. However, the answer will be reported as if it were an answer to an open query, simply because it has arguments.
- If a fact already present in a belief base is added again, as in the case where Mary repeats the information that the maid already gave, then that fact will appear twice in belief base on the screen. The GUI should refer to a set of beliefs rather than a list so that redundancy doesn't occur.

8.4 Cost Analysis Implementation

So that Mary could assign costs to her planned actions, I made some direct changes to the classes which implement intentions. The “Intention” class was given a field specifying the cost of executing the intention and a field specifying whether the intention was honest or a lie. These changes are also present in one of Intention's subclasses, the Skill class. The problem with these changes are that the cost and the honesty of an intention should depend on its context. To best fit with the agent-cycle model intended for Angerona, the action being considered should be taken from the context object associate with the intention in question, and then the intention update operator should determine the honesty of intention based on that action in that context. Rather than store the cost of executing an intention in the intention object, the cost would be stored elsewhere in the intention update operator class.

8.5 Internal Logic Comparisions

There are numerous cases where objects, belong to different types but both being used to represent logical statements, must be compared. Currently comparisons are made by parsing the string forms of the logical statements. For example, an answer is found for a question by checking whether the string form of the predicate of the fact matches that of the question. Instead a comparision between objects should be made, such as between the internal object representations of facts. Though this would ultimately come down to string comparision between the data held within the objects, it would be a cleaner solution. Likewise conversions between types should be done by building up the objects comprising those types rather than parsing the string form of the original type.

9 Outlook

I succeeded in modeling my scenario and having my agent “Mary” act according to her confidentiality policy. In doing so I discovered what confidentiality-preserving mechanisms from the background theory were necessary to include in the Angerona framework. As I included these mechanisms I realized further extensions which might be necessary for future simulations. In this section the most interesting of these further extensions are listed.

9.1 Further Extensions Relevant for Scenario

The following are suggested ways to build on what I did during my internship.

Further interpretation of d parameter

The d parameter used to quantify secrecy strength could also be used to quantify the credulity of a belief operator. One extreme of credulity, a totally credulous operator that believes a fact present in one answer set, could be specified with a d value of 1. On the other extreme, a totally skeptical operator which only believes a fact present in all answer sets, could be specified with a d value of 0. Of course, the value could be anywhere in between 0 and 1. The belief operator corresponding to a d value would be the most credulous operator possible for which a secret of strength d would not be revealed. For example, a secret of strength d=0 is only considered revealed when it is present in all answer sets, so the most credulous operator possible is the totally skeptical one. For a secret of strength d=1, the secret is considered revealed if it is present in one answer set. The totally credulous operator considers a secret safe only if it is present in no answer sets. Thus an operator with a d value of 1 is a totally credulous one. A d of 0.6 corresponds to an operator which considers a secret safe when it is present in less than 0.4 of answer sets. In other words, an operator with a d value of 0.6 considers a fact true if it is in 40% of answer sets.

Sources of information

Tagging sources of information would be a feature which could also prevent undesireable actions such as self-repeating. In addition, it could enable features such as sophisticated contradiction checking (discussed in the section below) or modeling trustworthiness.

Sources of information would ideally be stored in the beliefs of an agent. There, inference rules could be constructed to affect agent's behavior according to where facts or rules come from. For example, if the Coroner believed Mary was a pathological liar, he could have an inference rule to update his beliefs to the opposite of what Mary told him. Operators such as the belief revision operator could refer to the agent's beliefs for sources of information when deciding how to act. For example, in a model of trustworthiness, the Coroner could resolve conflicting information given to him by the maid and Mary by choosing to believe the information from the source he trusts more.

More sophisticated contradiction checking

Sometimes contradicting the beliefs of another agent is justifiable. For example, the Coroner might have mistaken information which Mary wants to correct. Mary could also learn new information regarding a topic which contradicts the old information she gave the Coroner, and she wants to share her new knowledge. For many situations it would suffice to have a distinction between creating a contradiction by truthful information and by telling a lie. In other situations, such as when Mary reasons the Coroner could believe her lies than another agent's truths, knowing the source of an agent's information would be useful.

Different belief change operator

Currently the belief revision operator used will prefer information coming from facts over information implied by rules. A more appropriate revision operator for my scenario would have the facts from rules stay and the contradictory facts get removed. This is because it is assumed that rules represent a set of reasoning processes which are more deeply rooted in an agent than facts. For this scenario that assumption is made because it is thought that real people trust their own reasoning processes better than they trust facts given to them by other people.

A further extension for Angerona would be to have a belief revision operator which allows some rules to be more deeply rooted than facts but not all rules. There could be some rules which are specified as deeply rooted while others would be overriden by newer information. Facts could likewise be specified as deeply-rooted or not. This could be a way to express beliefs in fundamental laws and facts, like the law of gravity and whatever rules of reasoning that entails. An ordering to how deeply-rooted a rule or a fact is could also be specified in the future.

Complementary Literals and Reasoning

When an agent possesses any reasoning operator where facts with a 50% probability or less are believed, there is the possibility that it has to choose its beliefs from contradictory answer sets. Suppose that an agent has two answer sets to consider, one containing a and one containing $\neg a$. Under a credulous operator the agent could believe either fact, but clearly not both. The operator could be described as “optimistic”, “pessimistic”, or “nihilistic”, and choose to either believe the positive fact, the negated fact, or neither, respectively. If a fact appears in more answer sets than its negation, the agent could be specified to believe the more frequently occuring fact. A parameter could specify at what minimum difference in frequency does the most frequent fact get favored and where otherwise its “optimism”, “pessimism”, and “nihilism” preferences take over.

9.2 Other Extensions Considered

Having questions update the views and beliefs of agents

By asking a question an agent reveals some information, but exactly what information has to be interpreted by the receiver of the question.

For example, if an agent asks “why were you at the cafeteria?” then that reveals he believes the other agent was at the cafeteria. There would be a classification of questions somewhere that explains what each type of question normally means.

A particularly useful application of tracking questions would be in inferring the reasoning operators of other agents. For example, let’s say that Mary believes the Coroner has a credulous reasoning operator. She knows that a piece of information revealed suggests “deep voice heard” in 50% of answer sets. She also believes from “deep voice heard” the Coroner can infer “John argued”. However, suppose the Coroner then asks her “Why did Alfred argue?”. She interprets this question as meaning the Coroner did not infer “John argued”, meaning she must either revise her beliefs of the Coroner’s inference rules or her belief that the Coroner is a credulous reasoner.

The asking of questions would be represented in a belief base the same way as a sources of information would be.

Updating agent views from world view

Sometimes new information about the world changes an agent’s beliefs about what other agents believe.

For a simple example, suppose agent Mary is wondering if agent Bob knows agent Tom was sneaking around his house. If Mary believes was_night, then she may have a rule representing Bob not knowing. If her belief gets updated to ¬was_night, then she may believe Bob does know. Views should thus be updated from rules in the world view.

Implementing this change in Angerona would be quite complicated. In fact, all the theory details aren’t worked out either.

Nevertheless it’s a change to consider.

10 Evaluation of the Project

The aim of the internship project was to model an application scenario for our prototype implementation of a confidentiality preserving multiagent system and to run experiments with that prototype as specified in Section 1. This aim has been completely satisfied within the time of the internship. Since the intern was an undergraduate with little previous knowledge about artificial intelligence and computer security a more theoretical approach to the matter of subject and work closer to the underlying theories was out of question. The intern already had to get acquainted with the BDI model, ASP, our framework for epistemic secrecy and the Angerona system.

An application scenario for our framework and implementation is very beneficial for our ongoing work not only to have a proof of concept but also to be able to model new aspects that have not been completely worked out theoretically, and to discover these new aspects in the first place. All of these benefits have been achieved by the internship project. The chosen scenario shows many aspects of confidentiality in a multiagent system and can be varied in many ways. At the time of the internship, some of those aspects were already covered by the theoretical work in project A5 or implemented in the Angerona system, yet, several aspects had not been under investigation so far. Hence, the intern had to develop and implement new parts in the Angerona system and was working on its own branch of it. We could observe that considered theoretical aspects were motivated independently by the chosen scenario such as the cost and the stability of lies, the weakening of secrets and the need for information about the sources of information. The new aspects disclosed in the scenario are in particular the need for open and detailed queries and the secrecy aspects and possibilities of deception by leaving out secrecy violating information when answering, and the need for advanced contradiction checking mechanisms based on the agent history including checks for self-repetition and contradiction. Moreover, the Angerona system was tested intensely during the internship project which led to many valuable improvements and bug-fixes.

Summing up, the internship was very valuable and successful on a series of levels. As just described it was successful in the sense that the intended goals were completely met and exceeded. Our previous and current work was confirmed and consolidated. The new aspects discovered already influence our ongoing and future work. Besides the technical level the project was valuable and fruitful for the intern and for the supervisors. For the intern it was a new challenge and experience to live and work abroad and in an environment completely new to him, with respect to the content as well as to the working environment. For its supervisors it was a way to broaden their experience to design such a project, to apply for the funding, to select an adequate candidate, and to take care of and to guide an intern from abroad in all matters involved during its three month stay.

References

[1] SFB 876 A5 project. Official project webpage. http://sfb876.tu-dortmund.de/SPP/sfb876-a5.html, 2011.

[2] Joachim Biskup. Inference-usability confinement by maintaining inference-proof views of an information system. *International Journal of Computational Science and Engineering*, 7(1):17–37, 2012.

[3] A. Christie. *The Mysterious Affair at Styles*. Hercule Poirot Mystery Series. 1st World Library, 2006.

[4] Michael Gelfond and Nicola Leone. Logic programming and knowledge representation — the A-Prolog perspective. *Artificial Intelligence*, 138(1–2):3–38, 2002.

[5] Patrick Krümpelmann and Gabriele Kern-Isberner. Agent-based epistemic secrecy. In *Proceedings of the 14th International Workshop on Non-Monotonic Reasoning (NMR12)*, 2012.

[6] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the 1st International Conference on Multiagent Systems (ICMAS'05)*, San Francisco, 1995.

[7] W. Spohn. Ordinal conditional functions: a dynamic theory of epistemic states. In W.L. Harper and B. Skyrms, editors, *Causation in Decision, Belief Change, and Statistics*, volume 2, pages 105–134. Kluwer Academic Publishers, 1988.

[8] G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1999.

11 Appendices

11.1 Appendix A: Background on Answer Set Programming

We give a short introduction to extended logic programs and the answer set semantics as presented in [4]. An extended logic program consists of rules over a set of atoms $\mathcal{A}$ using strong negation $\neg$ and default negation *not* . A literal L can be an atom $A \in \mathcal{A}$ or a negated atom $\neg A$. The complement of a literal L is denoted by $\neg L$ and is A if $L = \neg A$ and $\neg A$ if $L = A$. Let $\mathcal{A}$ be the set of all atoms and Lit the set of all literals $Lit = \mathcal{A} \cup \{\neg A \mid A \in \mathcal{A}\}$. For a set of literals $X \subseteq Lit$ we use the notation $not\ X = \{not\ L \mid L \in X\}$ and denote the set of all default negated literals as $\mathcal{D} = not\ Lit$. $\mathfrak{L} = Lit \cup \mathcal{D}$ represents the set of all literals and default negated literals. A rule r is written as

$$L \leftarrow L_0, \dots, L_m, not\ L_{m+1}, \dots, not\ L_n.$$

where the head of the rule $L = H(r)$ is either empty or consists of a single literal and the body $\mathcal{B}(r) = \{L_0, \dots, L_m, not\ L_{m+1}, \dots, not\ L_n\}$ is a finite subset of $\mathfrak{L}$. The language of rules constructed over the set of atoms $\mathcal{A}$ this way is referred to as $\mathcal{L}^{asp}_{At}$. A finite set of sentences from $\mathcal{L}^{asp}_{At}$ is called an extended logic program $P \subseteq \mathcal{L}^{asp}_{At}$.

The body consists of a set of literals $\mathcal{B}(r)^+ = \{L_0, \dots, L_m\}$ and a set of default negated literals denoted by $\mathcal{B}(r)^- = \{L_{m+1}, \dots, L_n\}$. Given this we can write a rule as

$$H(r) \leftarrow \mathcal{B}(r)^+, not\ \mathcal{B}(r)^-.$$

If $\mathcal{B}(r) = \emptyset$ we call r a fact. A set of literals that is consistent, i.e., it does not contain complementary literals L and $\neg L$, is called a state S. The reduct P^S of a program P relative to a state S is defined as

$$P^S = \{H(r) \leftarrow \mathcal{B}^+(r) \mid r \in P, \mathcal{B}^-(r) \cap S = \emptyset\}.$$

A state S is a model of a program without default negated literals P if for all $r \in P$ if $\mathcal{B}(r)^+ \subseteq S$, then $H(r) \cap S \neq \emptyset$. An answer set of a program P is a state S that is a minimal model of P^S. The set of all answersets of P is denoted by $AS(P)$.

Two different inference relations are defined based on answer sets. An extended logic program P infers a literal L credulously, denoted by $P \models^c_{asp} L$, iff $L \in \cup AS(P)$. An extended logic program P infers a literal L skeptically, denoted by $P \models^s_{asp} L$, iff $L \in \cap AS(P)$. $P \models^\circ_{asp} S$ refers to any answer set inference relation. The answer set semantics defines the evaluation of queries $?L$ with $L \in Lit$ as *yes* if $P \models^\circ_{asp} L$, *no* if $P \models^\circ_{asp} \neg L$ and *unknown* else.

12 Appendix B: Test Cases

In addition to my scenario's simulation and some experimental variations, I had to produce some simulations to test the features I added. Most test cases worked for the indended features and anticipated future uses (e.g. affecting more than one secret at once), but in a few cases bugs still occur. The following test cases provide a complete overview of the features added to Angerona as well as the known problems.

12.1 Core Scenario Test Cases

The following test cases describe the scenario I intended to model as well as some minor variations.

Full Scenario

Secret and strength: argued(john), d=0.5
Question: said(X) (two times)
Mary's beliefs:

```
world {
        said(youLied).
        said(youOwe).
        said(husbandWifeScandal).
        argued(john).
}
view->Coroner {
        said(husbandWifeScandal).
        argued(john) :- said(husbandWifeScandal), not argued(alfred).
        argued(alfred) :- said(husbandWifeScandal), not argued(john).
        argued(john) :- said(youOwe), not argued(alfred), not argued(lawrer
        argued(alfred) :- said(youOwe), not argued(john), not argued(lawrer
        argued(lawrence) :- said(youOwe), not argued(john), not argued(alfr

}
```

Output: The Coroner asks Mary "said(X)". Mary answers "said(husbandWifeScandal)". The Coroner asks her the same question. Mary answers "said(youLied)".

Mary has one secret – "argued(john)". The Coroner asks her "said(X)" twice. She knows of three things that were said, each of which affects the strength of her secret to a varying degree. The most secrecy-relevant piece of information, "said(husbandWifeScandal)" is already known by the Coroner (presumably revealed to him by another witness). Mary has to take this into account and weaken her secrecy accordingly. She can also say "dontKnow(said)", an act which she considers to have a fixed cost equal to weakening a secret by 0.5.

Full Scenario with Three Questions

Secret and strength: argued(john), d=0.5
Question: said(X) (three times)
Mary's beliefs: identical to beliefs in "Full Scenario"

Output: The Coroner asks Mary "said(X)". Mary answers "said(husbandWifeScandal)". The Coroner asks her the same question. Mary answers "said(youLied)". The Coroner asks Mary "said(X)" a third time. Mary answers "dontKnow(said)".

The same as the above scenario, except that the Coroner asks "said(X)" three times. Because “said(husbandWifeScandal)” is already known, the set of answer sets considered by the Coroner cannot be further reduced by new information. Thus further weakening of secrecy is not possible.

Full Scenario No Maid

Secret and strength: argued(john), d=0.7
Question: said(X) (two times)
Mary’s beliefs:

```
world {
        said(youLied).
        said(youOwe).
        said(husbandWifeScandal).
        argued(john).
} view->Coroner {

        argued(john) :- said(husbandWifeScandal), not argued(alfred).
        argued(alfred) :- said(husbandWifeScandal), not argued(john).
        argued(john) :- said(youOwe), not argued(alfred), not argued(lawrer
        argued(alfred) :- said(youOwe), not argued(john), not argued(lawrer
        argued(lawrence) :- said(youOwe), not argued(john), not argued(alfr

}
```

Output: The Coroner asks Mary “said(X)”. Mary answers “said(youLied)”. The Coroner asks her the same question. Mary answers “said(youOwe)”.

The same as the full scenario, except that the maid has not revealed “said(husbandWifeScandal)”. Mary’s secret is not weakened by that fact, and instead of her first priority revealing that fact is her last priority. She chooses to say “said(youLie)” and then “said(youOwe)” instead.

12.2 Feature Demonstration Test Cases

The following test cases highlight important features added to Angerona. The additions they highlight either were not included in the behavior of the full scenario or which were less obviously presented in the full scenario.

Full Scenario Two Secrets

Secret and strength: argued(john), d=0.7; eavesdropped_on_argument, d=1.0
Question: said(X) (two times)
Mary’s beliefs: identical to beliefs in “Full Scenario”

```
world {
        said(youLied).
        said(youOwe).
        said(husbandWifeScandal).
        argued(john).
        eavesdropped_on_argument.
} view->Coroner {

        argued(john) :- said(husbandWifeScandal), not argued(alfred).
        argued(alfred) :- said(husbandWifeScandal), not argued(john).
        argued(john) :- said(youOwe), not argued(alfred), not argued(lawrer
        argued(alfred) :- said(youOwe), not argued(john), not argued(lawrer
        argued(lawrence) :- said(youOwe), not argued(john), not argued(alfr
        eavesdropped_on_argument :- said(youOwe).

}
```

Output: The Coroner asks Mary “said(X)”. Mary answers “said(youLied)”. The Coroner asks her the same question. Mary answers “said(husbandWifeScandal)”. This case demonstrates Angerona’s ability to take two affect secrets into account. The scenario is the same as “Full Scenario No Maid”, but
“said(husbandWifeScandal)” is no longer the most costly piece of information to reveal. Instead, “said(youOwe)” is, because it completely reveals Mary’s secret “eavesdropped_on_argument”. Angerona adds the costs of weakening each secret affected. Mary responds to “said(X)” first with “said(youLie)” and then with “said(husbandWifeScandal)”.

Full Scenario with Five Questions

Secret and strength: argued(john), d=0.5
Question: said(X) (five times)
Mary’s beliefs: identical to beliefs in “Full Scenario”

Output: The Coroner asks Mary “said(X)”. Mary answers “said(youLied)”. The Coroner asks her the same question. Mary answers “said(youOwe)”. Coroner asks his question a third time. Mary answers “said(husbandWifeScandal)”. The Coroner asks Mary a fourth and fifth time, and Mary answers with “dontKnow(said)”. This case demonstrates that saying “dontKnow” is an exception to the no-repeating rule.

Blatant Contradiction with Secret

Secret and strength: john_argued, d=0.5
Question: john_argued
Mary’s beliefs:

```
world {
        john_argued.
} view->Coroner {
```

```
            -john_argued.
}
```

Output: The Coroner asks Mary “john_argued”. Mary answers “¬john_argued”

This case demonstrates agents’ ability to consider creating a contradiction the same as violating confidentiality.

Blatant Contradiction No Secret

Secret and strength: (none)
Question: said(X) (three times)
Mary’s beliefs:

```
world {
            john_argued
} view->Coroner {
            -john_argued.
}
```

Output: The Coroner asks Mary “john_argued”. Mary answers “john_argued” This case demonstrates that Angerona will not avoid creating contradictions when no secrets are at stake. The Coroner asks her the same question as in the previous scenario, but this time since Mary has no secret she wants to keep she answers with her belief “john_argued”.

Implicit Secrecy without Contradiction

Secret and strength: john_argued, 0.6
Question: deep_voice
Mary’s beliefs:

```
world {
            john_argued.
            deep_voice.
} view->Coroner {
            john_argued :- deep_voice.
}
```

Output: The Coroner asks Mary “deep_voice”. Mary answers “¬deep_voice”

This case demonstrates agents’ ability to protect secrets when asked for information that implies a secret, rather than a question about the secret itself. Mary’s secret is “john_argued”. The Coroner asks her “deep_voice”. She believes the Coroner has a rule “john_argued :- deep_voice”. When she simulates adding “deep_voice” to her view of the Coroner, “john_argued” becomes present as well. To protect her secret she answer “¬deep_voice”.

Implicit Secrecy with Contradiction

Secret and strength: john_argued, 0.6
Question: john_argued, deep_voice
Mary's beliefs:

```
world {
        john_argued.
        deep_voice.
} view->Coroner {
        john_argued :- deep_voice.
}
```

Output: The Coroner asks Mary "john_argued". Mary answers "¬john_argued". The Coroner asks "deep_voice". Mary answers "¬deep_voice".

This case demonstrates the most immediate need for agents to avoid contradictions. Mary has the secret "john_argued". The Coroner asks her "john_argued". She protects her secret by responding "john_argued=FALSE". Since Mary believes the Coroner believes everything she tells him, her view of the Coroner's beliefs now contains "¬john_argued". The Coroner then asks her "deep_voice". She believes "deep_voice" implies "john_argued" to the Coroner. Previously when she simulated adding "deep_voice" to her view of the Coroner, the contradiction between "¬john_argued" and "john_argued" was resolved by the belief revision operator by preferring the old fact, "¬john_argued". Thus the secret wasn't present and saying "deep_voice=TRUE" was considered safe. Removing the revision process from her consideration, however, leads to a contradictory answer set, which is represented a lack of any answer sets. There also the secret is not present and saying "deep_voice=TRUE" is considered safe. Only when a special check for contradictions – specifically a check for an empty answer set after a speech act – allows "deep_voice=TRUE" to be considered a violation of secrecy.

Negative Predicate

Secret and strength: (none)
Question: ¬said(X)
Mary's beliefs:

```
world {
        -said(youOwe).
        said(youLied).
        said(husbandWifeScandal).
}
view->Coroner {

}
```

Output: The Coroner asks Mary "¬said(X)". Mary answers "¬said(youOwe)"

This case demonstrates Angerona's current ability for recognizing negative predicates like "¬said(X)".

12.3 Bug Demonstration Test Cases

The following two test cases demonstrate some known problems with Angerona.

Arity 1 Closed Query

Secret and strength: (none)
Question: argued(john)
Mary's beliefs:

```
world {
            argued(john).
} view->Coroner {

}
```

Output: The Coroner asks Mary "argued(john)'. Mary answers "argued(john)" (problem lies in presentation of output)

Since a question like "argued(john)?" contains no ungrounded variables, it is rightly considered a closed query by the subgoal generation operator. However, due to how the output is determined, the output for an answer to a closed query with arguments will be treated the same as the answer to an open query. For this test case, rather than the answer "argued(john)=TRUE", the answer is "argued=john" (the format more appropriate for a question like "argued(X)").

Negative Secret

Secret and strength: (none)
Question: ¬alfred_argued
Mary's beliefs:

```
world {
            -alfred_argued.
}
view->Coroner {

}
```

Output: The Coroner asks Mary "¬alfred_argued". Mary doesn't answer – the program crashes

This test case demonstrates that it is not currently possible to keep secrets about negative facts, like "¬alfred_argued". In this test case the Coroner asks Mary "¬alfred_argued".

Mary believes "¬alfred_argued" but that is her secret. So she should answer "¬alfred_argued=FALSE". Instead, the program crashes after the Coroner asks her the question. The crash is because a signature was not assigned to the fact ¬alfred_argued, only alfred_argued, and the program considers ¬alfred_argued unrelated to alfred_argued rather than its negation.

13 Appendix C: Description and Pseudocode of Changes Made

The following sections describe the major changes made to the Angerona program. The violates, intention update, belief upate, reasoning, and subgoal generation operators were rewritten. Additionally, the new classes "LyingOperator" and "SecrecyStrengthPair" are included to better understand the algorithms of the other pseudocode. The changes made to the Agent, Intention, Skill, and BaseBeliefbase classes were only small additions. These changes and the descriptions of new helper classes like DetailQueryAnswer, DetailQueryDO, DetailQueryAnswerDO, and DetailPerceptionFactory are not described in pseudocode here because they exist only as support within the framework for the algorithms described in what was included.

13.1 C1 Subgoal Generation Operator

The file for this subgoal generation operator is called "SubgoalGenerationOperator" and is located in the "angerona.fw.mary" package.

Generating Ordered Questions

In method `interrogateOtherAgent`

```
Get all desires from object of agent in question
Sort desires in ascending order by number at the end of desire's name
```

Asking a Question

In method `interrogateOtherAgent`

```
Iterate through desires in reverse order:
  if name of desire doesn't start with a ``q\_'':
    continue
  predicateName :- portion of name before parentheses
  termNames :- portion of name within parentheses
  terms :- list of terms types.
   One element for each item in termNames string (items seperated by ``, '').
  query :- Detail Query object built from predName, terms
  subgoal :- new subgoal object
  add query to subgoal
```

Developing Options for Answers

In method `answerQuery`

```
Retrieve the query from the agent's perception
Retrieve from agent world views all facts related to query
 (as determined by agent's reasoning operator)
trueAnswers := list of these retrieved facts
//The first context must be made with a factory
//Others can be initialized from constructor
//Thus the first isn't made in the same loop as the others
Initialize new ``context'' object with agent's perception, using Context Factory class
subgoal := new subgoal object
add context to subgoal
Map string ``answer'' to first element in trueAnswers
Initialize new list allAnswers
Add each element in trueAnswers to allAnswers
pass query to ``simpleQuery'' method
if the query is simple:
  if trueAnswers has at least one element:
    truth := first element of trueAnswers
    lie := logical negation of truth
    add lie to allAnswers
add ``dontKnow('predName')'' to allAnswers
for all elements but first in allAnswers:
  elm := next unretrieved element
  context := new context object
  map string ``answer'' to elm
  add mapping to context
  add context to subgoal stack
add subgoal to plan
```

Determine Whether a Query is Closed

In method `simpleQuery`

```
if the arity of the query > 1:
  termList := list of terms from query
  if first term's name all upper case:
    return false
return true
```

13.2 C2 Intention Update Operator

The file for this intention update operator is called "MaryIntentionUpdateOperator" and is located in the "angerona.fw.operators.def" package.

Extracting Atomic Intentions

In method `processInt`

```
atomicIntentions := new empty list of intentions
for every subgoal in the agent's plans:
  for every stack of intentions in the subgoal:
    if the first element in the stack is an atomic action:
      intention := first element in stack
      if intention is a lie according to method isLie:
        assign cost to intention according to lyingCost
        add intention to atomicIntentions
      else:
        simulate running intention (using run method in intention object)
        skill := intention cast to skill object
        weakenings := weakenings field in skill
        assign cost to intention according to secrecyWeakeningsCost(weakenings)
        add intention to atomicIntentions
min := intention returned by minimalCostingIntention(atomicIntentions)
set agent's weakenings to min.getWeakenings()
return min
```

Assigning Cost to Lies

In method `lyingCost`

```
return estimate of 0.5
```

Determining Cost of Weakening Secrets

In method `secrecyWeakeningCost`

```
total := 0
for each SecrecyStrengthPair object in argument list:
  add pair.getDegreeOfWeakening() to total
return total
```

Choosing Minimal Costing Intention

In method `minimalCostingIntention`

```
minIntention := first element in argument list ``intentions''
minCost := cost associated with minIntention
for each intention in intentions:
  if intention cost less than minCost:
    minIntention := intention
    minCost := cost of intention
return minIntention
```

13.3 C3 Violates Operator

The file for this violates operator is called “WeakeningViolatesOperator” and is located in the “angerona.fw.operators.def” package.

Checking for Self-Repeating

In method `processIntAndWeaken`

```
a := answer being considered
secretList := empty list of SecrecyStrengthPair objects
for each action in agent's action history:
  if a equals action in a deep comparison (each field value compared):
    secretList := value returned by passing secretList to representTotalExposure method
```

Adding New Information to Simulated Program

In method `processIntAndWeaken`

```
view := deep copy of agent's view of answer's receiver
program := logic program stored in view
answerFormula := logic formula representation of a
rule := answerFormula converted to a logic rule
add rule to program
```

Checking for Contradictions

In method `processIntAndWeaken`

```
aspReasoner = reasoning operator stored in view
//The line below needs to be done to give the reasoner the expanded logic program of view
call infer method in aspReasoner, pass view
newAnsSets := list of answer sets from aspReasoner
if newAnsSets is null:
  //No answer sets (not even an empty one) means a contradiction occurred
  secretList := value returned by passing secretList to representTotalExposure
```

Determing Secrets Affected

In method `processIntAndWeaken`

```
secretContained := false
for each secret textit{curSecret} of the agent's secrets:
  secretInfo := logic formula representation of curSecret
  for each answer set textit{ans} in newAnsSets:
    if ans contains secretInfo:
      secretContained := true
  if secrectContained:
    sPair := new SecrecyStrengthPair
    define sPair's secret as curSecret
    curStrength := d value of curSecret
    newStrength := calculateSecrecyStrength(secretInfo, newAnsSets)
```

Determining Degree of Weakening

In method `calculateSecrecyStrength`

```
arguments: secretInfo (logic formula form of a secret) and ansSets (a list of answer sets)
setsWithSecret := 0
totalSets := number of sets in ansSets
for each answer set textit{as} in ansSets:
  program := answer set converted to logic program
  secretRule := secretInfo converted to a rule
  if the program contains secretRule:
    setsWithSecret++
quotient := setsWithSecret/totalSets
//Strength defined according to definition of d parameter
strength := 1 - quotient
return strength
```

Considering secrets totally exposed

In method `representTotalExposure`

```
list := list of SecrecyStrengthPair objects passed as an argument
for each element in list:
  set element's degreeOfWeakening  to 1000
return list
```

13.4 C4 Beliefs Update Operator

While the violates operator calculates the degree a secret would be weakened by an action, and the intention update operator chooses actions acocrdingly, it is up to the belief updates operator to also update the strength of secrets. The file for this beliefs update operator is called "MaryBeliefUpdateOperator" and is located in the "angerona.fw.operators.def" package.

Weakening actual secrets

In method `processInt`

```
//Recall that the weakenings field is set by the intention update operator when an action is chosen
weakenings := list of SecrecyStrengthPairs stored in the agent's object
for each secret textit{agentSecret} the agent holds:
  for each secret textit{weakenedSecret} present in an element of weakenings:
    if agentSecret matches weakenedSecret:
      //Remember that strength is quantified for a secret by its d value
      subtract from agentSecret's strength weakenedSecret's degree of weakening
```

13.5 C5 Reasoning Operator

The file for this reasoning operator is called "AspDetailReasoningOperator" and is located in the "angerona.fw.logic.asp" package.

Respond to an open query

In method `openQueryAnswers`

```
        Parameter: logical formula called query
        queryPredicate := predicate component of the query
        knowledge := list of logical formula forms of facts agent believes
        answer := empty list of logical formulas
        for each element f in knowledge:
          factPredicate := predicate component of f
          if factPredicate equals queryPredicate:
            add f to answers
        return answers
```

13.6 C6 LyingOperator

The file for this component is called "LyingModule" and is located in the "angerona.fw.mary" package. *In method* `lie`

```
    parameter: logical formula called truth
    if truth is a negation:
      return positive form of truth
    else:
      return negated form of truth
```

13.7 SecrecyStrengthPair

The file for this component is located in the "angerona.fw.logic" package. The class is so simple its entirety is represented below.

```
  secret := null
  degreeOfWeakening := 0
  setSecret(passedSecret):
    secret := passedSecret
  setDegreeOfWeakening(degree):
    degreeOfWeakening := degree
  getSecret():
    return secret
  getDegreeOfWeakening():
    return degreeOfWeakening
```