Reasoning about Agent's Private Knowledge

Michal Jakob, Jan Tožička, Michal Pěchouček, Olga Štěpánková
Gerstner Laboratory, Czech Technical University in Prague
{jakob|tozicka|pechouc|step}@labe.felk.cvut.cz
http://gerstner.felk.cvut.cz

Abstract

An important property of an autonomous agent is its ability to reason about other agents' private knowledge. Knowledge about the community and private knowledge of its members can be obtained by monitoring events appearing in the community. We propose three approaches how an agent can use the observed events to build such a model: one is based on theorem proving, the other on machine learning techniques, namely ILP and version space. The proposed framework allows to choose the type of model revision/update on eager–lazy strategy scale. These techniques were implemented and compared in two case studies concerning coalition formation.

1. Introduction

We are working with a community of autonomous reasoning agents endowed by number of capabilities which allow to form coalitions in order to cooperate in solving complex tasks. Behavior of our agents is given by a transparent reasoning algorithm (permanent while parameterized strategy) and a set of private knowledge describing e.g. agent's preferences concerning the tasks it wants to be involved in, preferred or unwanted locations, preferences of cooperation with other agents or participation in different type/size of coalitions. Agent's private knowledge is permanent. Dynamics of agent's behavior is given by changing resources and their availability, ever changing environment in which the agents operate or different behavior (in our scenario specified by a series of requests) of the other members of the multi-agent community.

What can also change or evolve during the lifespan of an agent is agent's awareness about the private knowledge of other members of the community. An agent has no direct access to the private knowledge of any other agent, it only can try to estimate or reconstruct its content. This estimate is very important in a number of tasks – it can influence complexity, quality and effectiveness of collaboration, as well as the response time of the system.

The main goal of our research was to suggest techniques and algorithms for identifying agent's private knowledge and decision making preferences. We have suggested a general meta-reasoning framework offering a choice of three knowledge processing techniques and experimented with a simple scenario.

Meta-reasoning is a key concept in this article. Unlike in classical computer science literature [8], where the meta-reasoning process is strictly understood as a reasoning process about yet another reasoning process, we will refer to meta-reasoning as agents capability to reason about other agents. Let us have a whole set of formulae that express some property of a specific agent (e.g. agent's resources, knowledge, plans, commitments, etc.). Meta-reasoning process is thus implemented by an algorithm that takes such a formula (one or many) as an input and produces another formula of this kind as an output.

In classical computer science meta-reasoning is not limited to mere reasoning about reasoning. It can also change the reasoning process that is being under investigation. Similarly, in multi-agent systems we distinguish between *passive* and *active* meta-reasoning. While the former only allows analysis of agents' behavior, the latter also implements a feedback to the community (e.g. once meta-reasoning process identifies an intruder, others shall be notified that there is an intruder in the community). Given the environment, we distinguish between collaborative an adversarial meta-reasoning. In the case of *collaborative* environment the agents are aware of being monitored, which is what they agree with and support. The purpose of collaborative meta-reasoning is very often an improvement of agents' collective behavior. In *non-collaborative* environment agents do not want to be monitored and do not support the meta-reasoning process. While in the former case the meta-reasoning agent may work

with e.g. copies of communicated messages, in the latter case more speculative techniques need to be used (e.g. community intrusion, or monitoring the environment [6]).

In principle, any agent can carry out meta-reasoning. Within this paper we have been investigating the situations with a single meta-reasoning agent that is busy exclusively with meta-reasoning process. This agent is referred to as meta-agent.

2. Scenario

The research is motivated by coalition formation problems introduced in the OOTW (Operations Other Than War) environment [17]. Let us consider a community of few dozens of agents, each providing some services and having a capability of requesting services from others. Agents are organized into **alliances** – semi-collaborative groups where they share some information about available resources [11]. This information is located in agents' acquaintance models. The agents are semi-collaborative as they work together on a mission while they are in the coalition, however if they are not committed to any collaboration pattern, they are self-interested.

Agents need resources in order to participate in solving some **mission** described by a set of properties (contains primarily a list of requested services. For the mission to be accomplished the corresponding services need to be implemented. The set of agents working on a mission is referred to as a **coalition**. The coalition is thus a collection of agents who commit themselves to participation in the mission by providing the requested services.

Once there is a request for mission operation, not necessarily all agents in the community are asked to form coalitions. Relevant agents are e.g. subscribed the scenario agent for notification about missions or chosen by scenario agent himself. Those agents, given the estimates of available resources of peer alliance members, construct **coalition proposal**. The coalition proposal consists of the list of possible coalition members, the overall objective function of the coalition (e.g. price, delivery date, ...) and the list of required services that cannot be provided from within the alliance.

After the coalition proposal is specified the agents enter a rather complicated negotiation process in order to fix a joint commitment that will ultimately form the needed coalition. The coalition that will cover a specific mission is constructed in three steps:

1. **Coalition Leader Selection.** Subscribed agents then inform each other about the value of the objective function of their coalition proposals and compete one another. Under an assumption that the agents are true telling, they allocate (using a simple bidding strategy) a **coalition leader** – an agent who covers the most within its own alliance (the most preferred criterion) and with the shortest delivery time.
2. **In-alliance Coalition Formation.** The coalition leader tries to form a part of the coalition from his alliance members. Given the knowledge in its acquaintance model, the coalition leader directly requests the agents for *(i)* participating in a mission that will take place in specific place and *(ii)* providing the required resources. While the coalition leader knows about resources availability, it is not aware of agents' private knowledge that may restrict it to work under certain agents' leadership (e.g. army unit) or a specific place (e.g. place with too small population).
3. **Inter-alliance Coalition Formation.** Coalition leader tries to subcontract – using the contract net protocol – agents from other alliances to contribute with services in order to cover the remaining requested services. In order to decrease the communication burden we will not want the coalition leader to do complete broadcasting. Instead only one member of each alliance (for simplicity we will refer in this report to this agent as a team-leader) is asked for the resources on behalf of the entire alliance.

Within the alliances the shared information is maintained by a subscribe-inform communication protocol.

2.1. Scenario Language

In order to present our research in more transparent way, let us briefly comment the specific scenario language. The scenario language comprises two types of constructs - *events* and *queries*.

2.1.1. Events. Events observed within the community are represented by *event formulas*, which consist of a conjunction of literals specifying the particular properties of the event:

$$\begin{aligned}&\texttt{accept}(\texttt{agent}_\texttt{A}, \texttt{c}_\texttt{id}) \wedge \texttt{mission}(\texttt{c}_\texttt{id}, \texttt{m}_\texttt{id}) \wedge \\ &\quad \wedge \texttt{location}(\texttt{m}_\texttt{id}, \texttt{location}) \wedge \texttt{size}(\texttt{c}_\texttt{id}, \texttt{size}) \wedge \\ &\quad \wedge \texttt{leader}(\texttt{c}_\texttt{id}, \texttt{agent}_\texttt{B}) \wedge \texttt{member}(\texttt{c}_\texttt{id}, \texttt{agent}_\texttt{i}) \wedge \\ &\quad \wedge \texttt{requirements}(\texttt{m}_\texttt{id}, \texttt{commodity}_\texttt{id}) \wedge \\ &\quad \wedge \texttt{amount}(\texttt{commodity}_\texttt{id}, \texttt{number})\end{aligned} \tag{1}$$

In this case, the predicate `accept` specifies whether the agent $\texttt{agent}_\texttt{A}$ would accept or reject a coalition $\texttt{c}_\texttt{id}$ working together in a specific mission $\texttt{m}_\texttt{id}$.

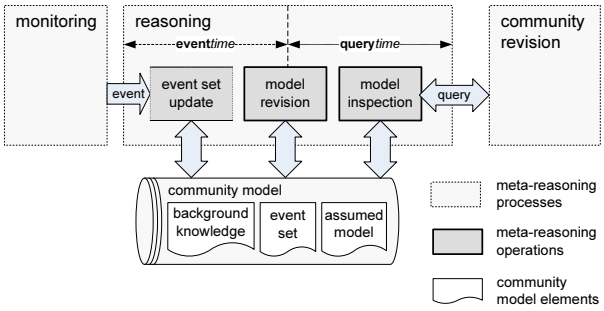

Figure 1. Meta-Reasoning Architecture Overview

Both the coalition and the mission can have various properties such as coalition size $\texttt{size}(\texttt{c}_{\texttt{id}}, \texttt{size})$, location $\texttt{location}(\texttt{m}_{\texttt{id}}, \texttt{location})$ or requirements specification. Similarly requests or providing information can be represented.

In this work, we limit ourselves to events that can be classified as either positive or negative. Thus, each event $\texttt{e}$ has an label $\oplus$ or $\ominus$ assigned. Acceptance/rejection of participation in a given coalition or mission is a typical example of $\oplus/\ominus$ type of events.

2.1.2. Queries. Queries posed to the meta-reasoning algorithm has the same form as event formulas. In addition, we also allow queries with quantified or uninstantiated variables. See below for an example:

$$\exists x \forall y \; \texttt{accept}(\texttt{agent}, \texttt{c}_{\texttt{id}}) \wedge \\ \wedge \texttt{mission}(\texttt{c}_{\texttt{id}}, y) \wedge \texttt{location}(y, x) \quad (2)$$

asking at which locations is the $\texttt{agent}$ happy to deliver.

3. Abstract Meta-Reasoning Architecture

The principal role of the meta-agent is to support meta-reasoning process through maintaining and exploiting a model of the agent community. In this work, we suppose that the strategy of agents does not change over the time. In principle, the community model can be maintained in two ways:

- **deductive reasoning** maintains the model so that it only contains formulas that logically follow from the monitored information (i.e. the model will be consistent with future possible events)
- **inductive reasoning** maintains an *approximative model* which also contains formulas generalizing the monitored information; this formula can prove to be in conflict with some future events.

Deductive reasoning produces a 'safe' model of the community. The price paid for this safety is that deductive reasoning cannot more often provide a definite *yes*/*no* answer to a query, and answers *unsure* instead. Thanks to its generalization capabilities, inductive reasoning returns definite answers in a greater number of situations. In some cases, however, it can provide false answers.

The architecture we have built to implement meta-reasoning is depicted in figure 1. Its building blocks are *meta-reasoning processes*, *meta-reasoning operations*, and the *community model*. In the following, we discuss each of them in more detail.

3.1. Meta-Reasoning Processes

Meta-reasoning process in multi-agent system comprises three mutually interconnected computational processes:

- **monitoring** process makes sure that the meta-agent knows the most it can get from monitoring the community of agents. Observed and recorded events serves as an input or a trigger for revision processes.
- **reasoning** process manipulates the model of the community so that true facts (other than directly observed) may be revealed; two key operations of the reasoning process are *model revision* and *model inspection*.
- **community revision** mechanism utilizes the community model (via the model inspection operation) to influence operation of the agent community; used exclusively within active meta-reasoning.

In the following, we will be almost exclusively concerned with the reasoning process.

3.1.1. Reasoning Process Phases. Regarding meta-agent interactions with the external environment, we can further distinguish two phases (we denote them as *times*) of the reasoning process:

- **event*time*** A new event $\texttt{e}$ in the community is reported to the meta-agent and is added to the model, resulting in an updated event set $\texttt{es}_n$. Depending on the meta-reasoning method, model revision operation can be carried out immediately or postponed to a later time.

- **query*time*** A meta-agent is queried about the community. Model inspection operation is performed and its result is returned to a client. Depending on the meta-reasoning method, model revision operation can be executed also prior to or even as a result of model inspection.

3.2. Community Model

The community model consists of three elements:

- background knowledge $\mathsf{bk}(\theta)$
- event set $\mathsf{es}(\theta)$
- assumed model $\underline{\mathsf{m}}(\theta)$

where θ is the set of agents which are the object of the reasoning. For the sake of simplicity, we omit the parameter θ in the following.

3.2.1. Background Knowledge. Background knowledge denoted as bk specifies relevant default properties of the agent community. Background knowledge is assumed to be always true and to be known to all agents before any event in the community happens. Background knowledge can have a number of different forms, while in our experiments we have been using the first-order predicate logic.

3.2.2. Event Set. Event set consists of event formulas representing events which were observed in the community thanks to meta-agent's monitoring capabilities. Thus, the event set is empty at the beginning of meta-reasoning process. When a new event e is observed, the corresponding event formula is added to the event set. We denote the event set as es, as or es_n in case we want to explicitly state how many events have been already stored. As already mentioned in Section 2.1, each event is labeled as positive or negative.

There are different types of events that the meta-agent is interested in. These include initiating a contract-net-protocol, sending a team allocation request, accepting or rejecting a team allocation request and informing about actual resources.

3.2.3. Assumed Model. Assumed model $\underline{\mathsf{m}}$ is the most dynamic component of the community model and it represents the current knowledgeabout the community. It is repetitively revised during the meta-reasoning process according to the observed events. The revision is carried out by model revision operator $\uplus$ during model revision, which is the key meta-reasoning operation.

3.3. Meta-Reasoning Operations

There are three operations each meta-reasoning method must implement.

3.3.1. Model Initialization. Model initialization is performed at the beginning of meta-reasoning process. The event set is cleared and the assumed model is set to an initial state as required by a given meta-reasoning method.

3.3.2. Model Revision. Model revision is the key operation within the meta-reasoning process. It is realized by the model revision operator $\uplus$ which generally takes the current model as the input and produces a new assumed model as an output, formally

$$\uplus \left\langle \mathsf{bk}, \mathsf{es}_n, \underline{\mathsf{m}}_{n-1} \right\rangle \rightarrow \underline{\mathsf{m}}_n \tag{3}$$

In case of incremental model revision which exploits only the last event, model revision can be viewed as follows

$$\uplus \left\langle \mathsf{bk}, \{\mathsf{e}_n\}, \underline{\mathsf{m}}_{n-1} \right\rangle \rightarrow \underline{\mathsf{m}}_n \tag{4}$$

Model revision usually takes place in **event***time*, but it can also occur during **query***time* or even during both – see Section 3.4 for details.

3.3.3. Model Inspection. Based on the client request, meta-agent queries the model and reports the result. Inspection is realized using the model inspection operator $\looparrowright$, which has three possible outcomes:

$$\looparrowright \langle \mathsf{bk}, \underline{\mathsf{m}}, \mathsf{q} \rangle \rightarrow \begin{cases} \mathit{yes} \\ \mathit{no} \\ \mathit{unsure} \end{cases} \tag{5}$$

An extended variant of the inspection operator can also treat queries with uninstantiated variables. In such situation, the reply can also contain possible values of the uninstantiated variables. Extended inspection operator thus allows to answer more complex queries rather than just *yes/no* queries.

For details concerning the implementation of the inspection operator, see 4. Model inspection occurs always in **query***time*.

3.4. Balancing Computational Requirements between event*time* and query*time*

Generally, reducing the time spent in the **event***time* increases the time spent in **query***time*. A meta-reasoning client can be thus interested in tuning the ratio between both times according to its preferences. The proper design depends on the required

meta-reasoning functionality. For visualization and intrusion detection the most of computation is required in the **event***time* since meta-reasoning outcomes are needed immediately. Event observation, model update and model inspection operation are closely integrated in this case. For simulation and prediction, on the other hand, the outcomes are only needed upon a client request. In this case, an important part of computational processes can be thus carried out in the **query***time* (for details see [13]).

3.4.1. Eager Revision/Lazy Revision. A general way to adjust the time ratio is by setting in which time of the reasoning process the model revision operation takes place. In case of *eager revision*, the community model is updated in **event***time*, i.e. immediately after a new event arrives. In case of *lazy revision*, any model updates are delayed until the **query***time*, i.e., until a client actually requests to inspect the model. The only operation performed in **event***time* is storing incoming observations for later processing. Lazy revision requires only a little time in **event***time* on the expense of longer **query***time* while for eager revision the situation is reversed. Except for these pure variants, a combination of lazy and eager revision is also possible.

3.4.2. Revision/Inspection. Sometimes the inspection and revision process can be designed so that the amount of computational resources required in one process directly influence the computational efficiency in the latter one. In such situation, increasing computational sophistication of the revision process can lead to faster and more straightforward inspection. This is the case with the parameterized model revision operator $\uplus_p^{\mathsf{TP}}$ in the theorem proving meta-reasoning method (see Section 4.1). It allows to adjust the time ratio between the **event***time* and **query***time* without relocating the model revision operation itself – model revision takes place completely in the **event***time*, only the level of its sophistication (a thus also time complexity) varies.

Depending on the meta-reasoning method, the ratio between both times can be adjusted in additional ways. One possibility is to focus the revision on those parts of the model which are most likely to be inspected in the **query***time*.

4. Meta-Reasoning Methods

4.1. Theorem Proving

Theorem proving, as a collection of problem solving techniques, has been widely used in symbolic artificial intelligence. Yet, this research domain remains an active field and new theorem provers are still being developed [14], [5]. For our experiments we have implemented a lightweight, resolution principle based reasoner [1]. The advantage of the implemented reasoner is its ability to deploy *parameterized revision operators*.

4.1.1. Parameterized Revision Operators. Theorem proving allows to introduce a parameterized version $\uplus_p^{\mathsf{TP}}$ of the model revision operator. Two extreme versions of parameterized revision operators are:

minimum revision operator $\uplus_{min}^{\mathsf{TP}}$ only appends an event formula to the assumed model. Formally,

$$\uplus_{min}^{\mathsf{TP}} \left\langle \mathsf{bk}, \mathsf{e}_n, \underline{\mathsf{m}}_{n-1}^{\mathsf{TP}} \right\rangle = \underline{\mathsf{m}}_{n-1}^{\mathsf{TP}} \cup \{\mathsf{e}_n\} \qquad (6)$$

maximum revision operator $\uplus_{max}^{\mathsf{TP}}$ revises the model so that it contains all[1] facts that logically follow from the original model $\underline{\mathsf{m}}^{\mathsf{TP}}$ combined with the new event e_n. Formally,

$$\uplus_{\max}^{\mathsf{TP}} \left\langle \mathsf{bk}, \mathsf{e}_n, \underline{\mathsf{m}}_{n-1}^{\mathsf{TP}} \right\rangle = \\ = \left\{ \varphi \mid \mathsf{bk} \cup \underline{\mathsf{m}}_{n-1}^{\mathsf{TP}} \cup \{\mathsf{e}_n\} \vdash \varphi \right\} \qquad (7)$$

where φ is a formula of first-order predicate logic .

The actual revision operator $\uplus_p^{\mathsf{TP}}$ resides somewhere in between these extreme variants. For models obtained by application of the parameterized revision operator it thus holds

$$\uplus_{\min}^{\mathsf{TP}} \left\langle \mathsf{bk}, \mathsf{e}, \underline{\mathsf{m}}^{\mathsf{TP}} \right\rangle \subseteq \\ \subseteq \uplus_p^{\mathsf{TP}} \left\langle \mathsf{bk}, \mathsf{e}, \underline{\mathsf{m}}^{\mathsf{TP}} \right\rangle \subseteq \\ \subseteq \uplus_{\max}^{\mathsf{TP}} \left\langle \mathsf{bk}, \mathsf{e}, \underline{\mathsf{m}}^{\mathsf{TP}} \right\rangle \qquad (8)$$

The parametrization of $\uplus_p^{\mathsf{TP}}$ can be done in a number of ways. Parameter p can specify eg. the maximum length formulas considered in (7), maximum depth of resolution used to deduce new formulas or the size of a subset of the original model on which $\uplus_{\max}^{\mathsf{TP}}$ operator is applied. Regardless of the actual parametrization used, decreasing p should move the $\uplus_p^{\mathsf{TP}}$ operator towards the minimal version $\uplus_{\min}^{\mathsf{TP}}$ and vice versa.

The motivation for introducing parameterized revision operator $\uplus_p^{\mathsf{TP}}$ is as follows: the lower p is, the faster should be the revision operation (since fewer formulas have to be deduced and added to the model) while

1 In many cases, the $\uplus_{\max}^{\mathsf{TP}}$ operator cannot be implemented as the resulting model would be large or even infinite – we introduce it primarily as an abstract boundary concept.

the inspection operation can take longer (since longer proofs are necessary to determine whether a goal is provable). This relation, however, does not hold for high values of p when the revision operator approaches the extreme version $\uplus^{\mathsf{TP}}_{\max}$. Even if we introduce some constrains (such as maximal length of a formula) in order to avoid infinite sets, the revised model could become voluminous. Searching such a model may be more time demanding than proving a respective query from a compact theory.

Nevertheless, at least for a certain range of parameter values, the parameterized revision operator $\uplus^{\mathsf{TP}}_{p}$ provides a mean to adjust the ratio between the time spent in model revision and model inspection, and consequently between the time spent in **event***time* and **query***time*. For experimental results concerning different values of parameter p, see Section 5.

4.1.2. Theorem Proving Meta-Reasoning Operations.

The individual operations of theorem proving meta-reasoning method are as follows.

model initialization $\underline{\mathsf{m}}^{\mathsf{TP}}_0$ is initialized to contain no element.

mode revision Updated model

$$\underline{\mathsf{m}}^{\mathsf{TP}}_n := \uplus^{\mathsf{TP}}_p \left\langle \mathsf{bk}, \mathsf{e}_n, \underline{\mathsf{m}}^{\mathsf{TP}}_{n-1} \right\rangle \tag{9}$$

is constructed using the $\uplus^{\mathsf{TP}}_p$ operator.

model inspection for a given model $\underline{\mathsf{m}}^{\mathsf{TP}}_n$ and query q, the inspection operator is defined as follows

$$\looparrowright^{\mathsf{TP}} \left\langle \mathsf{bk}, \underline{\mathsf{m}}^{\mathsf{TP}}_n, \mathsf{q} \right\rangle \to \\ \to \begin{cases} yes, & \text{if } \mathsf{bk} \cup \underline{\mathsf{m}}^{\mathsf{TP}}_n \vdash \mathsf{q} \text{ is provable;} \\ no, & \text{if } \mathsf{bk} \cup \underline{\mathsf{m}}^{\mathsf{TP}}_n \vdash \neg\mathsf{q} \text{ is provable;} \\ unsure, & \text{otherwise.} \end{cases} \tag{10}$$

4.2. Version Space

Version space algorithm [9] uses the least expressive hypothesis space – its hypotheses are conjunctive clauses in propositional logic. Because of the limited expressivity, version space algorithm can reason only about individual agents, not about entire communities as is the case with theorem proving and inductive logic programming methods.

The VS algorithm represents the space of all hypotheses by two sets: $\mathcal{G}$ – set of the most general consistent hypotheses and $\mathcal{S}$ – set of the most specific consistent hypotheses, so that all and only those hypotheses, that are more general than some hypothesis in $\mathcal{S}$ and more specific than some hypothesis in $\mathcal{G}$ are consistent with processed events. Therefore the model managed by VS algorithm is a pair:

$$\underline{\mathsf{m}}^{\mathsf{VS}} = \langle \mathcal{S}, \mathcal{G} \rangle \tag{11}$$

model initialization The set $\mathcal{G}$ contains the most general hypothesis (always responding *yes*) and the set $\mathcal{S}$ contains the most specific hypothesis.

model revision Incoming positive event results in generalization of the $\mathcal{S}$ set and removing the event from the $\mathcal{G}$. Analogically for negative events.

model inspection

$$\looparrowright^{\mathsf{VS}} \left\langle \mathsf{bk}, \underline{\mathsf{m}}^{\mathsf{VS}}_n, \mathsf{q} \right\rangle \to \\ \to \begin{cases} yes, & \text{if all hypotheses in } \mathcal{G} \text{ and } \mathcal{S} \\ & \text{evaluate the } \mathsf{q} \text{ as positive;} \\ no, & \text{if all hypothese in } \mathcal{G} \text{ and } \mathcal{S} \\ & \text{evaluates the } \mathsf{q} \text{ as negative;} \\ unsure, & \text{otherwise.} \end{cases} \tag{12}$$

Background knowledge bk is not explicitly used during inspection.

4.3. Inductive Logic Programming

Inductive logic programming (ILP) [10] has already been successfully applied in multi-agent systems [7]. Its main advantage is the ability to learn from structured data and to incorporate background knowledge in the learning process.

In ILP meta-reasoning method, the assumed model consists of a pair of first-order theories[2]:

$$\underline{\mathsf{m}}^{\mathsf{ILP}} = \left\langle \underline{\mathsf{m}}^{\mathsf{ILP}\oplus}, \underline{\mathsf{m}}^{\mathsf{ILP}\ominus} \right\rangle \tag{13}$$

where $\underline{\mathsf{m}}^{\mathsf{ILP}\oplus}$ approximates the set of positive and $\underline{\mathsf{m}}^{\mathsf{ILP}\ominus}$ of negative events. $\underline{\mathsf{m}}^{\mathsf{ILP}\oplus}$ can be viewed as a set of first-order-logic rules which – in conjunction with the background knowledge – cover the positive events, analogically for $\underline{\mathsf{m}}^{\mathsf{ILP}\ominus}$.

model initialization Both models are set to be empty.

model revision A revised pair of models

$$\left\langle \underline{\mathsf{m}}^{\mathsf{ILP}\oplus}_n, \underline{\mathsf{m}}^{\mathsf{ILP}\ominus}_n \right\rangle = \uplus^{\mathsf{ILP}} \left\langle \mathsf{bk}, \mathsf{es}_n, \underline{\mathsf{m}}^{\mathsf{ILP}}_{n-1} \right\rangle \tag{14}$$

is obtained by applying an ILP Aleph [16] system as follows:

$$\underline{\mathsf{m}}^{\mathsf{ILP}\oplus}_n := \mathsf{ILP} \left\langle \mathsf{bk}, \mathsf{es}_n, \underline{\mathsf{m}}^{\mathsf{ILP}\oplus}_{n-1} \right\rangle \tag{15}$$

$$\underline{\mathsf{m}}^{\mathsf{ILP}\ominus}_n := \mathsf{ILP} \left\langle \mathsf{bk}, \overline{\mathsf{es}_n}, \underline{\mathsf{m}}^{\mathsf{ILP}\ominus}_{n-1} \right\rangle \tag{16}$$

2 Precisely, first-order Horn theories

where $\overline{\mathsf{es}_n}$ contains the same events as es_n but with inverted labels (i.e. with positive and negative events interchanged);

model inspection Two queries are issued to the Prolog interpreter concerning both the positive and the negative model. Result of the inspection is defined as follows.

$$\looparrowright^{\mathsf{ILP}} \left\langle \mathsf{bk}, \underline{\mathsf{m}}^{\mathsf{ILP}}, \mathsf{q} \right\rangle \rightarrow$$
$$\rightarrow \begin{cases} \textit{yes}, & \text{if } \mathsf{bk} \wedge \underline{\mathsf{m}}^{\mathsf{ILP}\oplus} \vdash \mathsf{q} \text{ is provable} \\ & \text{and } \mathsf{bk} \wedge \underline{\mathsf{m}}^{\mathsf{ILP}\ominus} \vdash \mathsf{q} \text{ is not provable;} \\ \textit{no}, & \text{if } \mathsf{bk} \wedge \underline{\mathsf{m}}^{\mathsf{ILP}\oplus} \vdash \mathsf{q} \text{ is not provable} \\ & \text{and } \mathsf{bk} \wedge \underline{\mathsf{m}}^{\mathsf{ILP}\ominus} \vdash \mathsf{q} \text{ is provable;} \\ \textit{unsure}, & \text{otherwise.} \end{cases} \tag{17}$$

In other words, the answer is positive if the query is covered by the positive rules and not covered by the negative rules and vice versa. The answer is *unsure* if both rule sets are in conflict or if none of them cover the query.

5. Experiments

To evaluate meta-reasoning methods, we have used two existing scenarios developed in our lab. Both of them are based on coalition planning agents. The three implemented methods were integrated in the meta-reasoning agent that operates in the CPlanT coalition formation multi-agent system [11] and two methods (ILP and theorem proving) were integrated in $\mathcal{A}$-CROSS multi-agent system. The goal of the meta-reasoning exercise was to reconstruct agents decision making algorithm.

5.1. CPlanT Scenario

In the CPlanT system, the problem was simplified so that all the agents have used the same decision making algorithm with different collaboration preferences. The meta-reasoning agent was employed to identify these preferences – an instance of private knowledge (e.g. never work with partners form the specific country, always prefer coalitions under governmental coordination, etc.). For full description of the domain and agent's decision making algorithm see [1]. A large number of experiments has been performed in order to verify used technologies, algorithms and proposed improvements.

We have been working with a background knowledgeconsisting of 249 formulae. An event set of the specific missions has been defined and sent to the CPlanT

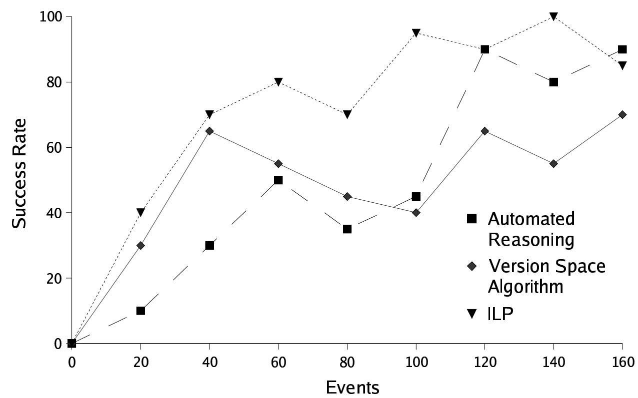

Figure 2. Comparison of Selected Meta-Reasoning Methods in CPlanT Scenario: Evolution of Success Rates

multi-agent system. The meta-reasoning agent tried to predict whether the object agent will accept or refuse a possible team allocation request.

See the graph in figure 2 for studying how well the meta-reasoning methods performed at the time of learning. The vertical axis specifies the number of events predicted successfully. The complement to these numbers says how many times the method replied *unsure*. The exception is the ILP method, that gave also 3 % of incorrect prediction (i.e. replied *yes* instead of *no* or otherwise). The horizontal axis gives the number of events that happened in the community (and the size of training set). ILP meta-reasoning operators outperform the classical theorem proving technique and version space algorithm. An interesting observation is that VS operates well with small number data – up to 60 events, and with time if performs much worse than ILP. Resolution based meta-reasoning operator does not perform very well with small number of data while with about 140 events it provides good performance.

The table 1 illustrates how well did the meta-reasoning methods performed after the learning phase. The number in the table denotes how many definite answers were given in the query phase.

5.1.1. Parameterized Model Revision Operators Using theorem proving as a meta-reasoning method we can implement parameterized $\uplus$ model revision operators as defined in Section 4.1.1. We will denote them as $\uplus_p^{\mathsf{TP}}$, where p indicates the number of background-knowledge formulae (from total of 249 formulae) in **event***time* used to deduce all *relevant* consequences of new event e_n. The choice

Resolution	Version Space	ILP
54 %	54 %	79 %

Table 1. Comparison of Selected Meta-Reasoning Methods in CPlanT : Average Success Rates

Model Revision Operator	Generated Clauses	
	Model Revision	Model Inspection
$\uplus^{\mathsf{TP}}_{\min}$	0	119498
$\uplus^{\mathsf{TP}}_{7}$	451	96451
$\uplus^{\mathsf{TP}}_{73}$	655	74927
$\uplus^{\mathsf{TP}}_{90}$	821	74864
$\uplus^{\mathsf{TP}}_{235}$	1553	12183

Table 2. Theorem Proving: Comparison of Parameterized Model Revision Operators.

of ground belief formulae is driven by a threshold of complexity – this process is described in details in [1]. Having the scale of $\uplus^{\mathsf{TP}}_{p}$ operators we can choose the best one for our meta-agent. Generally, it holds that the more queries come per one event the closer should be the chosen $\uplus^{\mathsf{TP}}_{p}$ operator to $\uplus^{\mathsf{TP}}_{\max}$, and vice versa. Because more complex process in **event***time* can reduce the time response in **query***time*.

In this experiment, we have compared following revision operators: $\uplus^{\mathsf{TP}}_{\min}, \uplus^{\mathsf{TP}}_{7}, \uplus^{\mathsf{TP}}_{73}, \uplus^{\mathsf{TP}}_{90}, \uplus^{\mathsf{TP}}_{235}$. Table 5.1.1 shows the number generated clauses in the **event***time* and in the **query***time*.

This experiment shows that the choice of revision operator $\uplus^{\mathsf{TP}}_{p}$ can significantly reduce the process in **query***time*. But in the case, that the model contains large number of formulae, **query***time* can be negatively affected by searching and maintaining this model.

5.2. $\mathcal{A}$-cross Scenario

Agents in the **$\mathcal{A}$-cross** system [12] have different decision making algorithms that is not known to meta-agents. Under such a condition it was not possible to use version space algorithm anymore as the structure of decision making algorithm has to be encoded into update and revision operators. Theorem proving has used this knowledge as its background knowledge. Without having this knowledge, it works as simple database only as it is not able to deduce any new knowledge.

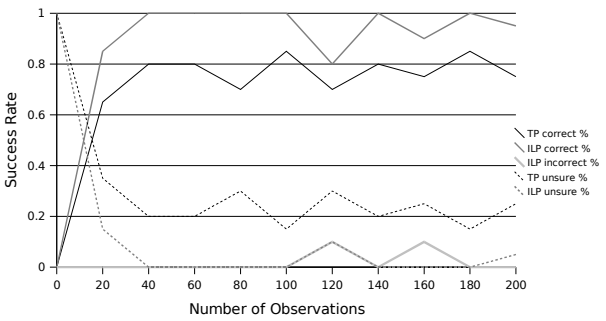

Figure 3. Comparison of Selected Meta-Reasoning Methods in $\mathcal{A}$-cross Scenario: Evolution of Success Rates

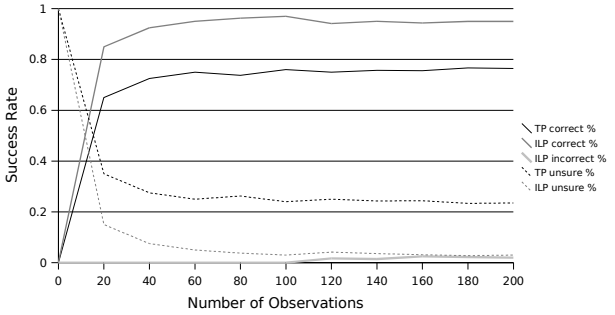

Figure 4. Comparison of Selected Meta-Reasoning Methods in $\mathcal{A}$-cross Scenario: Life-Time Success Rates

In figure 3 we can see that both methods have been more successful than in CPlanT scenario. It is caused by different coalition formation strategies of planning agents that leads to repetitive generation of same events (same coalitions are created for same missions). The success rate is measured in the same way as in the figure 2. ILP method had better success rate but it also gave us 4 false predictions.

Figure 4 shows average success rates of methods during the whole run of meta-agent.

The table 3 illustrates how well did the meta-reasoning methods performed after the learning phase. The number in the table denotes how many definite answers were given in the query phase.

Theorem Proving	ILP
77 %	95 %

Table 3. Comparison of Selected Meta-Reasoning Methods in $\mathcal{A}$-cross : Average Success Rates

This experiment leads to the similar conclusions as in CPlanT scenario even though the distributions of events have been different.

In both scenarios, the inductive logic programming method seems to reach the best results due to the ability to generalize knowledge via the induction operation. On the other hand ILP gave several incorrect predictions. There are no incorrect predictions if we use the automated reasoning method based on the deduction operation. The version space method (used in the CPlanT scenario only) does not generate incorrect predictions although the induction operation is used here, because it stores all possible hypothesis consistent with the previous events. The results of the automated reasoning (without the support of the reasoning simulation) and version space methods are similar. The version space method uses the same background knowledge as the automated reasoning method. While the automated reasoning method uses background knowledge in explicit form, version space method uses background knowledge in an implicit form (encoded into the hypothesis space structure).

6. Conclusions

We have carried out a series of experiments with three meta-reasoning techniques in two scenarios. Neither machine learning nor theorem proving has proofed to be perfect solution to private knowledge detection exercise. Both the resolution and version space has been outperformed by ILP in the number of correct predictions while it gave us also several false predictions.

Version space is a computationally efficient technique with a simple revision phase and also a comparatively fast inspection phase. The difficulty with version space is that it is very inexpressive. It does not support reasoning e.g. about collection of agents, their relations or sequences of events, or their more complex decision making algorithms. This was the reason why the version space algorithm could not be used in $\mathcal{A}$-CROSS domain.

Expressiveness of theorem proving and ILP is similar. Given its inductive capabilities, ILP may however provide more sophisticated generalizations resulting in far better inspection performance. The cost we have to pay for this is twofold: Firstly, an ILP based meta-reasoning mechanism may give false answers when queried about previously unseen events. This may be an obstacle in life critical applications where misclassification is not allowed. Secondly, the current implementation uses *non*-incremental ILP algorithm used for model revision which has high computational requirements. Model inspection, on the other hand, is quite fast. This makes ILP more suitable to applications where frequency of queries is substantially higher than frequency of new observations, or where the lazy revision strategy is suitable. This includes scenarios where meta-reasoning is used for explanation or prediction tasks. However, as long as very fast responses are required (e.g. visualization and intrusion detection) ILP is inappropriate. Using theorem proving algorithm we can balance its complexity in **event***time* and **query***time* to get optimal solution for meta-agent task.

The described research exercise has been constrained exclusively to a single meta-reasoning agent. Alternatively we may have a **team of meta-reasoning agents** that monitor different groups of agents (e.g. defined geographically), monitor the community behavior at different times, or monitor different aspect of agent's functionality. An appropriate mechanisms for knowledge fusion [2] need to be designed.

Even more challenging task is to provide all the agents in community with meta-reasoning capability. Agents would be able to reason about the others in addition to their *primary functionality* (planning, negotiation, diagnostic, etc.). This make the domain very different to the single meta-reasoning agent situation.

Firstly, agents would be required to reason not only about the other agents in isolation but about the behavior of the other agents in relation to the behavior of the agent itself. More importantly, the meta-reasoning algorithms would need to be extremely lightweight so that the primary functionality will be affected as little as possible. At the same time, on-line meta-reasoning in such a situation needs to be faster than the changes in the community and environment (this is not the case of the off-line meta-reasoning). This requirement does not depend only on the frequency of the changes in the community but also on the complexity of the meta-reasoning tasks. In most real-life application areas the meta-reasoning task is nontrivial, and the frequency of changes in the environment (or in the community) tends to be high. Non-incremental ILP and theorem proving seems inappropriate for on-line peer-

to-peer meta-reasoning.

In [4] it has been shown that ILP can be successfully applied to extend the BDI agent architecture with agents' learning skills. This approach has been tested in domain where the agents are learning conditions for excitability of their plans. Based on the research presented, we believe that ILP has got an important potential for implementing meta-reasoning in multi-agent systems.

Even though very simple behavior can be modeled by first-order predicate logic, in realistic scenarios the model needs to be extended by several types of modalities. The possibility of adding such modalities is explored mainly in the automated reasoning domain: *modal logics*, *dynamic logics*, *temporal logics* [15],[3]. While the theoretical background of these systems is well investigated, they have not been much used because of the complexity of prove searching process. Nevertheless, these problems are important research challenges for logic based meta-reasoning in the future.

7. Acknowledgment

The project work has been co-funded by European Office for Aerospace Research and Development (EORD) Air Force Research Laboratory (AFRL) – contract number: FA8655-02-M-4056, Office of Naval Research (ONR) – award number: N00014-03-1-0292 and by the Grant No. LN00B096 of the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] J. Bárta, J. Tožička, M. Pěchouček, and O. Štěpánková. Meta-reasoning in CPlanT multi-agent system. Technical Report GL 166/03. 56 p. ISSN 1213-3000, `http://agents.felk.cvut.cz/papers/MRinMAS.pdf`, Gerstner Laboratory ©, Czech Techical University in Prague, July 2003.

[2] B. Bell, E. Santos, and S. M. Brown. Making adversary decision modeling tractable with intent inference and information fusion. In *Proc. of the 11th Conf on Computer Generated Forces and Behavioral Representation*, Orlando FL., 2002.

[3] M. Fisher. A survey of concurrent metatem - the language and its applications. In D. Gabbay and H. J. Ohlbach, editors, *Temporal Logic - Proceedings of the 1st International Conference*, LNAI 827, pages 480–505. Springer, 1994.

[4] A. Guerra-Hernandez, F.-S. A., and H. Soldano. Learning in BDI multi-agent systems. In *in Proceedings of CLIMA 2003*, pages 185–200, 2004.

[5] J. Kalman. *Automated Reasoning with Otter*. Rinton Press, Inc., Prinecton, New Jersey, 2001.

[6] G. Kaminka, D. Pynadath, and M. Tambe. Monitoring deployed agent teams. In *Proceedings of the fifth international conference on Autonomous agents*, Montreal, Quebec, Canada, 2001.

[7] D. Kazakov and D. Kudenko. Machine learning and inductive logic programming for multi-agent systems. In *Multi-Agent Systems and Applications*, volume 2086 of *Lecture Notes in Artificial Intelligence*, pages 246–270, Prague, Czech Republic, 2001. Springer Verlag.

[8] P. Maes. Computational reflection. Technical report 87-2, Free University of Brussels, AI Lab, 1987.

[9] T. M. Mitchell. *Machine Learning*, pages 29–36. McGraw-Hill, New York, 1997.

[10] S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[11] M. Pěchouček, V. Mařík, and J. Bárta. A knowledge-based approach to coalition formation. *IEEE Intelligent Systems*, 17(3):17–25, 2002.

[12] M. Pěchouček, M. Rehák, M. Rollo, D. Šišlák, and J. Tožička. Solving coordination inaccessibility in coalition operations. In *Knowledge Systems for Coalition Operatioins 2004*, pages 19–36. CTU, Prague, 2004.

[13] M. Pěchouček, O. Štěpánková, V. Mařík, and J. Bárta. Abstract architecture for meta-reasoning in multi-agent systems. In Mařík, Muller, and Pěchouček, editors, *Multi-Agent Systems and Applications III*, number 2691 in LNAI. Springer-Verlag, Heidelberg, June 2003.

[14] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Communications*, 15(2-3), 2002.

[15] M. P. Singh, A. S. Rao, and M. P. Georgeff. *Multiagent Systems A Modern Approach to Distributed Artificial Intelligence*, chapter Formal Methods in DAI: Logic Based Representation and Reasoning, pages 201–258. MIT Press, Cambridge, MA., 1999.

[16] A. Srinivasan. The Aleph Manual. `http://web.comlab.ox.ac.uk/oucl/ /research/areas/machlearn/Aleph/`, 2003.

[17] E. C. T. Walker. Coalition planning for operations other than war. Technical report, Panel Report: Workshop at AIAI, Edinburgh, 1999.