Meta-Agent Programs

Jürgen Dix* V.S. Subrahmanian† George Pick‡

September 9, 1998

Abstract

There are numerous applications where one agent a needs to reason about the beliefs of another agent, as well as about the actions that other agents may take. Eiter, Subrahmanian, and Pick (1998) introduced the concept of an agent program, and provided a language within which the operating principles of an agent could be declaratively encoded on top of imperative data structures. We first introduce certain belief data structures that an agent needs to maintain. Then we introduce the concept of a *Meta Agent Program* (map), that extends the (Eiter, Subrahmanian, and Pick 1998) framework, so as to allow agents to peform metareasoning. We build a formal semantics for maps, and show how this semantics supports not just beliefs agent a may have about agent b's state, but also beliefs about agents b's beliefs about agent c's actions, beliefs about b's beliefs about agent c's state, and so on. Finally, we provide a translation that takes any map as input and converts it into an agent program such that there is a one-one correspondence between the semantics of the map and the semantics of the resulting agent program. This correspondence allows an implementation of maps to be built on top of an implementation of agent programs.

1 Introduction

Over the last few years, there has been tremendous interest in the area of intelligent software agents. Such agents provide a wide range of services, ranging from providing data mediation agents (Bayardo *et. al.* 1997; Arens, Chee, Hsu, and Knoblock 1993; Brink, Marcus, and Subrahmanian 1995; Lu, Nerode, and Subrahmanian 1996; Chawathe, Garcia-Molina, Hammer, Ireland, Papakonstantinou, Ullman, and Widom 1994), to mobile agents (Rus, Gray, and Kotz 1997), to personalized visualization agents (Candan, Prabhakaran, and Subrahmanian 1996; Ishizaki 1997), to agents that monitor newspapers, prioritize mail buffers and the like (Goldberg, Nichols, Oki, and Terry 1992; Foltz and Dumais 1992; Sta 1993; Sheth and Maes 1993).

*University of Koblenz-Landau, Dept. of Computer Science, Rheinau 1, D-56075 Koblenz, Germany. E-mail: dix@informatik.uni-koblenz.de.

†Institute for Advanced Computer Studies, Institute for Systems Research and Department of Computer Science, University of Maryland, College Park, Maryland 20742. E-mail: vs@cs.umd.edu

‡Department of Computer Science, University of Maryland, College Park, Maryland 20742. E-mail: george@cs.umd.edu

Most such existing work on agents subscribes to the view that agents should be *autonomous*, and that such autonomous agents should behave according to a clearly articulated set of *operating principles.* These operating principles allow agents to take actions that change the state of the agent in accordance with the operating principles (Rosenschein (1985), Rosenschein and Kaelbling (1995)). Declarative languages to encode such operating principles were proposed by Shoham(Shoham 1993) and Hindriks, de Boer, van der Hoek, and Meyer (1997). Recently, Eiter, Subrahmanian, and Pick (1998) have proposed a notion of an *Agent Program* and shown how agent programs can be layered on top of arbitrary data structures. This allows the creator of an agent to agentize existing bodies of software code by "adding on" such operating principles on top of the code.

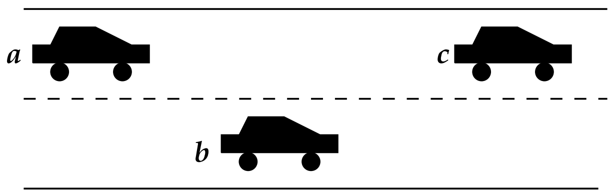

Figure 1: Autonomous Vehicle

In general, in many multiagent applications, an agent needs to be able to reason about other agents. Consider, for example, a simple autonomous vehicle $\mathfrak{a}$ driving in the left lane of a highway, with another vehicle in the right lane, as shown in Figure 1. Vehicle $\mathfrak{b}$ is in the right lane, slightly ahead of Vehicle $\mathfrak{a}$.

- Vehicle $\mathfrak{a}$ may believe that vehicle $\mathfrak{b}$'s turn signal is malfunctioning. This belief may well be incorrect.
- Vehicle $\mathfrak{a}$ may also believe that vehicle $\mathfrak{b}$ is imminently going to cut in front of it.

Based on these two beliefs—the first which is about the state of vehicle $\mathfrak{b}$, while the second is about the actions of vehicle $\mathfrak{b}$—vehicle $\mathfrak{a}$ may decide to slow down by either depressing the brake or easing up on the accelerator. Of course, things can get even more complex—for example, vehicle $\mathfrak{a}$ may believe that vehicle $\mathfrak{b}$ believes that vehicle $\mathfrak{c}$ is about to shift to the right lane, etc.

In general, this very simple, everyday example shows that one agent may need to reason about other agents' state, beliefs, and potential actions. In most existing agent languages such as (Shoham 1993; Hindriks, de Boer, van der Hoek, and Meyer 1997; Eiter, Subrahmanian, and Pick 1998), the notion of an agent state is general enough to include almost anything. However, the specific needs of agents such as vehicle $\mathfrak{a}$ in the above discussion, are not addressed, and in particular, it is assumed that beliefs will somehow be encoded into the very general notion of state. Our aim in this paper is to *precisely show* how this can be done, and we go about it in the following way:

- In Section 2, we present a compelling motivating example, that requires meta-reasoning capabilities.

- This paper forms part of the *IMPACT* (*Interactive Maryland Platform for Agents Collaborating Together*) project (Arisha, Kraus, Ozcan, Ross, and V.S.Subrahmanian 1997; Eiter, Subrahmanian, and Pick 1998; Bonatti, Kraus, Salinas, and Subrahmanian 1998). In Section 3, we briefly overview the basic architecture of our *IMPACT* system, and quickly describe the decision making framework of (Eiter, Subrahmanian, and Pick 1998).

- The new contributions of this paper start with Section 4. Here, we note that different agents may wish to reason about beliefs in different ways. Some may be content with reasoning about their beliefs about other agents, instead of their beliefs about the beliefs of other agents, and so on. We propose a hierarchy of belief languages, building on top of arbitrary data structures that embody the state of an agent. We then propose two specific data structures for managing an agent's beliefs—a *belief table*, and a *belief semantics table*. These are accompanied by corresponding operators to manipulate these tables. We then introduce the important notion of a Meta Agent Program (map, for short).

- In Section 5, we study the semantics of maps. We propose a notion of a *feasible belief status set*. Intuitively, a set of beliefs that satisfies these feasibility requirements is one that a "sensible" agent could hold. We refine this semantics to two finer grained semantics—namely *rational* belief status sets, and *reasonable belief status sets*, both of which satisfy additional epistemic requirements.

- In Section 6, we provide a transformation that takes any map, and converts it into an ordinary agent program, together with some integrity constraints, as defined by Eiter, Subrahmanian, and Pick (1998). A somewhat complex result shows that the *feasible* (resp. *rational*, *reasonable*) belief status sets of the map are in one-one correspondence with the *feasible* (resp. *rational*, *reasonable*) status sets (without beliefs) of the agent program + integrity constraints generated by the transformation. As techniques to implement agent programs have been undergoing concurrent development (see `http://www.cs.umd.edu/~vs/agent/impact.html` for selected screendumps), this means that once this transformation is implemented, maps can be computed in much the same way as agent programs' *feasible*, *rational*, and *reasonable* status sets.

2 Motivation: Route and Maneuver Planning (**RAMP**)

We are building an application to conduct distributed simulations involving route and maneuver planning over free terrain (RAMP). A simplified version of the RAMP application that deals with meta-reasoning by agents is described below. This example will provide a unifying theme throughout this paper, and will be used to illustrate the various definitions we introduce.

The RAMP application involves tracking enemy vehicles on the battlefield, and attempting to predict what these enemy agents are likely to do in the future, based on metaknowledge that we have about them. RAMP is intended to be used in training and simulation efforts, rather than being deployed on the battlefield. RAMP involves the following agents.

A set of enemy vehicle agents: These agents move across free terrain, and their motion is generated by a program that the other agents listed below do not have access to (though they may have beliefs about this program).

A terrain route planning agent, which reasons with terrain maps stored in the form of *Digital Terrain Elevation Data* (*DTED*). The terrain route planning agent takes as input, any *DTED* map, together with two points on this map, a vehicle type, and plans an optimal route from the first to the second point for the specified vehicle type. It also provides a flight path computation service for helicopters, through which it plans a flight, given an origin, a destination, and a set of constraints specifying the height at which the helicopters wish to fly. The terrain route planning agent is built on top of an existing *US ARMY Route planning software package* developed at the *Topographic and Engineering Center* (Benton and Subrahmanian 1994).

A tracking agent, which takes as input, a *DTED* map, an `id` assigned to an enemy agent, and a time point, and produces as output, the location of the enemy agent at the given point in time (if known) as well as its best guess of what kind of the enemy agent is.

A coordination agent, that keeps track of current friendly assets. This agent receives input and ships requests to the other agents with a view to determining exactly what target(s) the enemy columns may be attempting to strike, as well as determining how to nullify the oncoming convoy. The situation is complicated by the fact that the agent may have a hard time determining what the intended attack target is. It may be further complicated by uncertainty about what kind of vehicle the enemy is using—depending upon the type of vehicle used, different routes may be designated as optimal by the terrain route planning agent.

A set of helicopter agents, that may receive instructions from the coordination agent about when and where to attack the enemy vehicles. When such instructions are received, the helicopter agents contact the terrain route planning agent, and request a flight path. Such a flight path uses terrain elevation information (to ensure that the helicopter does not fly into the side of a mountain).

The aim of all agents above (except for the enemy agents) is to attack and nullify the enemy attacking force. To do this, the coordination agent sends requests for information and analyses to the other friendly agents, as well as instructions to them specifying actions they must take. It is important to note that the coordination agent's actions are based on its *beliefs* about what the enemy is likely to do. These beliefs include:

- *Beliefs about the type of enemy vehicle type.* Each enemy vehicle has an associated type—for example, one vehicle may be a T-80 tank, the other may be a T-72 tank. However, the coordination agent may not precisely know the type of a given enemy vehicle, due to inaccurate and/or uncertain identification made by the sensing agent. At any point in time, it holds some beliefs about the identity of enemy vehicle.
- *Beliefs about intentions of enemy vehicle.* The coordination agent must try to guess what the enemy's target is. Suppose the tracking agent starts tracking a given enemy agent at time t_0, and the current time is t_{now}. Then the tracking agent can provide information about the location of this agent at each instant between time t_0 and time

t_{now}. Let ℓ_i denote the location of one such enemy agent at time t_i, $0 \le i \le now$. The coordination agent believes that the enemy agent is trying to target one of its assets $A_1, \ldots, A_k$, but does not know which one. It may ask the terrain route planning agent to plan a route from ℓ_0 to each of the locations of $A_1, \ldots, A_k$, and may decide that the intended target is the location whose associated route most closely matches the observed initial route taken by the enemy agent between times t_0 and t_{now}.

- *Changing beliefs with time.* As the enemy agent continues along its route, the coordination agent may be forced to revise its beliefs, as it becomes apparent that the actual route being taken by the enemy vehicle is inconsistent with the expected route. Furthermore, as time proceeds, sensing data provided by the tracking agent may cause the coordination agent to revise its beliefs about the enemy vehicle type. As the route planning agent plans routes based on the type of enemy vehicle being considered, this may cause changes in the predictions made by the terrain planning agent.

- *Beliefs about the enemy agent's reasoning.* The coordination agent may also hold some beliefs about the enemy agents' reasoning capabilities (see the *Belief-Semantics Table* in Definition 4.7 on page 19). For instance, with a relatively unsophisticated and disorganized enemy whose command and control facilities have been destroyed, it may believe that the enemy does not know what moves friendly forces are making. However, in the case of an enemy with viable/strong operational command and control facilities, it may believe that the enemy does have information on the moves made by friendly forces—in this case, additional actions to mislead the enemy may be required.

A detailed description of all agents and their actions will be given in the Appendix B.

3 Preliminaries

For the rest of the paper we denote by A a finite set whose elements are called *agents*. Each agent $\mathfrak{a} \in \mathsf{A}$ is built on top of a body of software code (built in any programming language) that supports a well defined application programmer interface (either part of the code itself, or developed to augment the code). In general, we will assume that the piece of software $\mathcal{S}^{\mathfrak{a}}$ associated with an agent $\mathfrak{a} \in \mathsf{A}$ is represented by a pair $\mathcal{S}^{\mathfrak{a}} = (\mathcal{T}^{\mathfrak{a}}_{\mathcal{S}}, \mathcal{F}^{\mathfrak{a}}_{\mathcal{S}})$ where:

- $\mathcal{T}^{\mathfrak{a}}_{\mathcal{S}}$ is the set of all data types manipulated by the software package $\mathcal{S}^{\mathfrak{a}}$.

- $\mathcal{F}^{\mathfrak{a}}_{\mathcal{S}}$ is the set of all pre-defined functions of the package $\mathcal{S}^{\mathfrak{a}}$ that are provided by the package's application programmer interface.

When we are referring to the code associated with a fixed agent $\mathfrak{a}$, we will often drop the superscript $\mathfrak{a}$ above.

This characterization of a piece of software code is a well accepted and widely used specification. For example, the *Object Data Management Group*'s *ODMG* standard (Cattell *et.al.* 1997) and the *CORBA* framework (Siegal 1996) are existing industry standards consistent with this specification.

In our framework, agents will be built on top of a body of code (either legacy code, or specially developed code) satisfying the above definitions. Each agent has a message box (which will be discussed later in Example 3.8 on page 11) having a well defined set of associated code calls that can be invoked by external programs. Each agent has some metaknowledge about itself, as well as about other agents, reflecting its beliefs about the data possessed by other agents, the mechanism that other agents use to act, and the capabilities of other agents. Each agent has a security module that specifies the agent's associated security mechanisms, if any—the security module of our system is described in detail in (Bonatti, Kraus, Salinas, and Subrahmanian 1998).

The state of an agent, at any given point t in time, consists of the set of all instantiated data objects of types contained in $\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}$, as well as the contents of the metaknowledge module and the security module.

Each agent has an action-base consisting of a description of the various actions that the agent is capable of executing. Actions change the state of the agent and perhaps the state of other agents' `msgboxes`. Each agent has an associated set of integrity constraints—only states that satisfy these constraints are considered to be *valid* or *legal* states. Each agent has an associated set of action constraints that define the circumstances under which certain actions may be concurrently executed. As at any given point t in time, many sets of actions may be concurrently executable, each agent has a *Meta-Agent Program* (`map` for short and denoted by $\mathcal{BP}$), to be introduced in Definition 4.13 on page 24, that determines what actions the agent can take, what actions the agent cannot take, and what actions the agent must take. The `map` associated with an agent is a declarative specification of the agents' decision policies.

Figure 2 on the next page shows the different components of an agent, together with information on the flow of data/actions between them. The shaded components of this figure show objects whose contents jointly describe the state of the agent. The primary aim of this paper is to describe the meta-knowledge component of an agent's state, as well as the notion of a meta-agent program based on meta-knowledge.

3.1 Code Calls and Code Call Atoms

Suppose we consider a body $\mathcal{S} = (\mathcal{T}_{\mathcal{S}}, \mathcal{F}_{\mathcal{S}})$ of software code. Given any type $\tau \in \mathcal{T}_{\mathcal{S}}$, we will assume that there is a set $Var(\tau)$ of variable symbols ranging over τ. If $\mathtt{X} \in Var(\tau)$ is such a variable symbol, and if τ is a complex record type having fields $\mathtt{f}_1, \ldots, \mathtt{f}_n$, then we require that $\mathtt{X.f_i}$ be a variable of type τ_i where τ_i is the type of field $\mathtt{f_i}$. In the same vein, if $\mathtt{f_i}$ itself has a sub-field g of type γ, then $\mathtt{X.f_i.g}$ is a variable of type γ, and so on. In such a case, we will call $\mathtt{X}$ a *root-variable*, and the variables $\mathtt{X.f_i}$, $\mathtt{X.f_i.g}$, etc. *path-variables*. For any path variable $\mathtt{Y}$ of the form $\mathtt{X.path}$, where $\mathtt{X}$ is a root variable, we refer to $\mathtt{X}$ as the root of $\mathtt{Y}$, denoted by $root(\mathtt{Y})$; for technical convenience, $root(\mathtt{X})$, where $\mathtt{X}$ is a root variable, refers to itself.

3.1 Definition (Code Call cc)
Suppose $\mathcal{S} = (\mathcal{T}_{\mathcal{S}}, \mathcal{F}_{\mathcal{S}})$ is some software code and $f \in \mathcal{F}$ is a predefined function with n arguments, and $\mathtt{d}_1, \ldots, \mathtt{d}_n$ are objects or variables such that each $\mathtt{d_i}$ respects the type requirements of the i'th argument of f. Then $\mathcal{S}:f(\mathtt{d}_1, \ldots, \mathtt{d}_n)$ is called a code call. *A code call is* ground, *if all the $\mathtt{d_i}$'s are objects.*

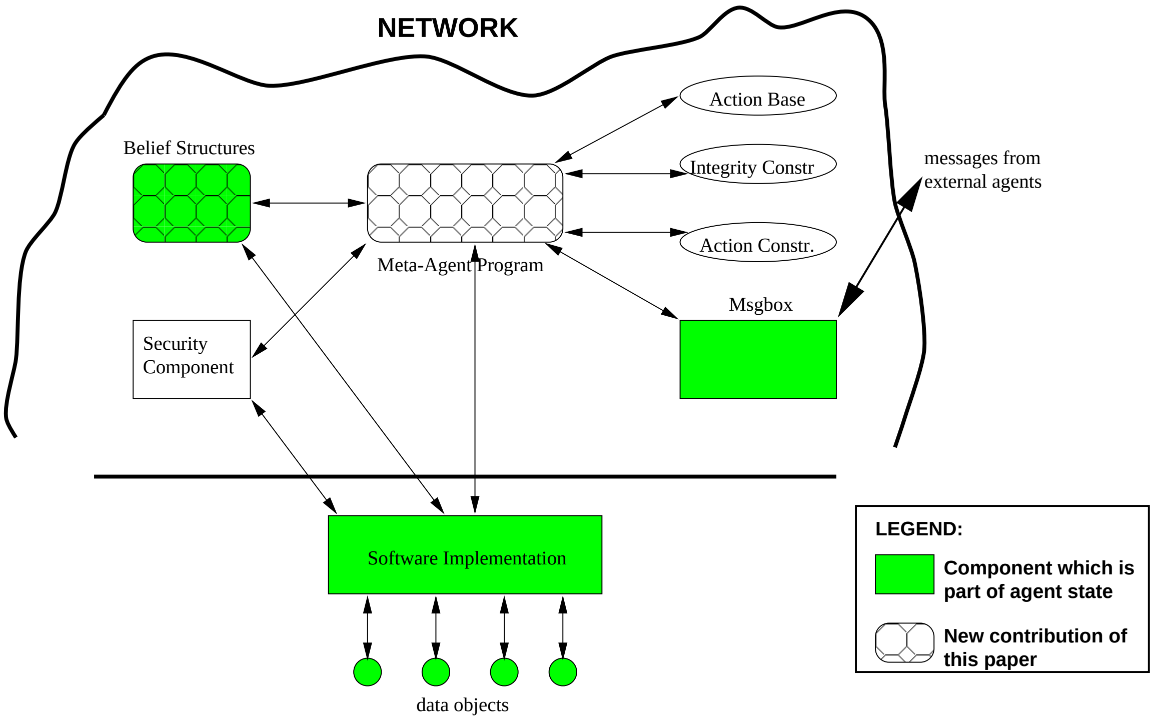

Figure 2: Overall Architecture

In general, as we will see later, code calls are executable when they are ground. Thus, non-ground code calls must be *instantiated* prior to attempts to execute them.

In general, each function $f \in \mathcal{F}$ has a *signature*, specifying the types of inputs it takes, and the types of outputs it returns. Here are some examples of code calls that are used in the RAMP example:

- $\mathtt{Heli}: \mathit{SetAltitude}(\mathtt{Alt}) \rightarrow \mathtt{Bool}$.

  This code call sets helicopter altitude to `Alt`. It returns **true** if it succeeds, otherwise it returns **false**.

- $\mathtt{Tank}: \mathit{GetPosition}(\mathtt{now}) \rightarrow \mathtt{2DPoint}$.

  This code call determines the current position of the tank.

- $\mathtt{Route}: \mathit{FlightPlan}(\mathtt{SourcePoint}, \mathtt{DestinationPoint}) \rightarrow \mathtt{SequenceOf3DPoints}$.

  This code call creates a flight plan from point `SourcePoint` to point `DestinationPoint`. It returns the plan as a sequence of 3D points.

- $\mathtt{Track}: \mathit{GetTypeOfAgent}(\mathtt{AgentId}) \rightarrow (\mathtt{VehicleType}, \mathtt{Probability})$.

  This code call determines the type of a vehicular agent whose id is `AgentId`. It also returns a number, denoting the probability that the identification is current.

- $\mathtt{Coord}: \mathit{FindAttackTimeAndPosition}(\mathtt{AgentId}) \rightarrow (\mathtt{Position}, \mathtt{Time}, \mathtt{Route}, \mathtt{Probability})$.

This code call tries to create a plan to attack agent with id $\mathtt{AgentId}$. The plan is returned as a position, a time point, a route to get there and the probability that the determination was correct.

3.2 Definition (Code Call Atom in(X, cc))
If $\mathtt{cc}$ *is a code call, and* $\mathtt{X}$ *is either a variable symbol, or an object of the output type of* $\mathtt{cc}$*, then* $\mathbf{in}(\mathtt{X}, \mathtt{cc})$ *and* $\mathbf{not\_in}(\mathtt{X}, \mathtt{cc})$ *are* code call atoms.

3.3 Definition (Code Call Condition χ)
A code call condition χ *is defined as follows:*

1. *Every code call atom is a code call condition.*
2. *If* $\mathtt{s}, \mathtt{t}$ *are either variables or objects, then* $\mathtt{s} = \mathtt{t}$ *is a code call condition.*
3. *If* $\mathtt{s}, \mathtt{t}$ *are either integers/real valued objects, or are variables over the integers/reals, then* $\mathtt{s} < \mathtt{t}, \mathtt{s} > \mathtt{t}, \mathtt{s} \geq \mathtt{t}, \mathtt{s} \leq \mathtt{t}$ *are code call conditions.*
4. *If* χ_1, χ_2 *are code call conditions, then* $\chi_1 \,\&\, \chi_2$ *is a code call condition.*

A code call condition satisfying any of the first three criteria above is an atomic *code call condition.*

The following is an atomic code call condition. It is satisfied if the position $\mathtt{pos}$ is the current position for the tank agent: $\mathbf{in}(\mathtt{pos}, \mathsf{Tank}: \mathit{GetPosition}(\mathtt{now}))$. Here is a more complex code call condition.

$$\begin{array}{ll}
\mathbf{in}(\mathtt{P2}, \mathsf{Heli}: \mathit{GetPosition}(\mathtt{now})) & \& \\
\mathbf{in}(\mathtt{D}, \mathsf{Heli}: \mathit{ComputeDistance}(\mathtt{P1}, \mathtt{P2})) & \& \\
\mathbf{in}(\mathtt{R}, \mathsf{Heli}: \mathit{GetMaxGunRange}(\mathtt{now})) & \& \\
\mathtt{D} < \mathtt{R} &
\end{array}$$

If $\mathtt{P1}$ is instantiated, then this code call attempts to check if the helicopter is within firing range of an enemy site located at $\mathtt{P1}$.

3.4 Definition (Safe Code Call)
A code call $\mathcal{S}: f(\mathtt{d_1}, \ldots, \mathtt{d_n})$ *is* safe *if, by definition, each* $\mathtt{d_i}$ *is ground. A code call condition* $\chi_1 \,\&\, \ldots \,\&\, \chi_n$*,* $n \geq 1$ *is* safe*, if, by definition, there exists a permutation* π *of* $\chi_1, \ldots, \chi_n$ *such that for every* $i = 1, \ldots, n$ *the following holds:*

1. *If* $\chi_{\pi(i)}$ *has the form* $\mathtt{s} = \mathtt{t}$ *or* $\mathtt{s} < \mathtt{t}$, $\mathtt{s} \leq \mathtt{t}$, $\mathtt{s} > \mathtt{t}$, $\mathtt{s} \geq \mathtt{t}$*, then one of* $\mathtt{s}$, $\mathtt{t}$ *(or both) is either a constant or one of the* $\mathtt{X}_{\pi(\mathtt{j})}$*'s for* $\mathtt{j} < \mathtt{i}$*; let* $\mathtt{X}_{\pi(\mathtt{i})}$ *denote a possible new variable;*
2. *If* $\chi_{\pi(i)}$ *is a code call atom* $\mathbf{in}(\mathtt{X}_{\pi(\mathtt{i})}, \mathtt{cc}_{\pi(\mathtt{i})})$ *or* $\mathbf{not\_in}(\mathtt{X}_{\pi(\mathtt{i})}, \mathtt{cc}_{\pi(\mathtt{i})})$*, then for each variable* $\mathtt{Y}$ *occurring in* $\mathtt{cc}_{\pi(i)}$*,* $\mathit{root}(\mathtt{Y})$ *is from the set* $\{\mathit{root}(\mathtt{X}_{\pi(\mathtt{j})}) \mid j < i\}$*.*

It is easily seen that the code call condition immediately preceding this definition is *not* safe. The reason for this is that the variable $\mathtt{P1}$ is not instantiated by any of the **in** atoms. Had $\mathtt{P1}$ been replaced with a ground term, then the above code call condition would have been safe.

3.2 Integrity Constraints $\mathcal{IC}$

In addition to code-calls, each agent also has an associated set of *Integrity Constraints*. Agent integrity constraints specify properties that states of the agent must satisfy.

3.5 Definition (Integrity Constraints $\mathcal{IC}$)
An integrity constraint *is an expression of the form*

$$\psi \;\Rightarrow\; \chi$$

where ψ is a safe code call condition, and χ is an atomic code call condition such that every root variable in χ occurs in ψ. A set of integrity constraints is denoted by $\mathcal{IC}$.

Here are two examples:

$$\Rightarrow \;\; \mathbf{in}(\mathtt{S}, \mathsf{Tank}: \mathit{GetSpeed}(\mathtt{now})) \quad \& \;\mathtt{S} < \mathtt{MaxSpeed}$$
$$\Rightarrow \;\; \mathbf{in}(\mathtt{A}, \mathsf{Heli}: \mathit{GetAltitude}(\mathtt{now})) \quad \& \;\mathtt{A} < \mathtt{MaxAltitude}$$

The first integrity constraint says that a tank's speed can never exceed its maximum speed, while the second says that a helicopter's altitude can never exceed its maximum flying altitude. In both the above examples, the ψ component of the integrity constraint is empty.

3.3 Actions

Every agent's actions are completely determined by three parameters that the individual creating the agent must specify:

$\mathcal{AB}$: an **Action Base**, specifying a set of actions that the agent can execute (under the right conditions),

$\mathcal{AC}$: a set of **Action Constraints** that specify, for example, mutual exclusion between actions, etc.

$\mathcal{P}$: a **Agent Program** that determines which of the (instances of) actions in the agent base the agent is obligated, permitted, or forbidden to execute, together with a mechanism to actually determine what actions will be taken.

3.3.1 Action Base $\mathcal{AB}$

In this section, we will introduce the concept of an action and describe how the effects of actions are implemented.

3.6 Definition (Action; Action Atom)
An action α *consists of five components:*

Name: *usually written $\alpha(\mathtt{X}_1, \ldots, \mathtt{X}_n)$, where the $\mathtt{X}_i$'s are root variables.*

Schema: *usually written as* $(\tau_1, \dots, \tau_n)$, *of types. Intuitively, this says that the variable* $\mathtt{X}_i$ *must be of type* τ_i, *for all* $1 \leq i \leq n$.

***Pre*(α):** *A code-call condition* χ, *called the* precondition *of the action.*

***Add*(α):** *A set* $Add(\alpha)$ *of code-call conditions.*

***Del*(α):** *A set* $Del(\alpha)$ *of code-call conditions.*

The precondition $Pre(\alpha)$ *must be* safe modulo the variables $\mathtt{X}_1, \dots, \mathtt{X}_n$. *This means that* $Pre(\alpha)$ *is a safe code-call condition if every variable* $\mathtt{Y}$ *in* $Pre(\alpha)$ *such that* $root(\mathtt{Y}) \in \{\mathtt{X}_i \mid 1 \leq i \leq n\}$ *were considered as an instantiated object (constant) from the domain. Furthermore, every code-call condition* χ *in* $Add(\alpha) \cup Del(\alpha)$ *must be safe modulo the union of* $\mathtt{X}_1, \dots, \mathtt{X}_n$ *and the root variables* $\mathtt{Y}_1, \dots, \mathtt{Y}_m$ *occurring in* $Pre(\alpha)$, *i.e., it is safe if every variable* $\mathtt{Y}$ *in* χ *such that* $root(\mathtt{Y}) \in \{\mathtt{X}_1 \dots, \mathtt{X}_n, \mathtt{Y}_1, \dots, \mathtt{Y}_m\}$ *were considered as though it were a constant.*

An action atom *is a formula* $\alpha(t_1, \dots, t_n)$, *where* t_i *is a term, i.e., an object or a variable, of type* τ_i, *for all* $i = 1, \dots, n$.

Let us now consider some examples of action and their associated descriptions in the case of the RAMP example described at the beginning of this paper.

3.7 Example (Some Actions of RAMP Agents)
We describe some actions of the Helicopter-, Tank-, Route- and Coordination-Agents:

Helicopter Agent:

$\mathit{Fly}(\mathtt{From}, \mathtt{To}, \mathtt{Altitude}, \mathtt{Speed})$
$\mathtt{Pre:}$ $\mathbf{in}(\mathtt{From}, \mathrm{Heli}: \mathit{GetPosition}(\mathtt{now}))$
$\mathtt{Del:}$ $\mathbf{in}(\mathtt{From}, \mathrm{Heli}: \mathit{GetPosition}(\mathtt{now}))$
$\mathtt{Add:}$ $\mathbf{in}(\mathtt{To}, \mathrm{Heli}: \mathit{GetPosition}(\mathtt{now}+1))$

Tank Agent:

$\mathit{Drive}(\mathtt{From}, \mathtt{To}, \mathtt{Speed})$
$\mathtt{Pre:}$ $\mathbf{in}(\mathtt{From}, \mathrm{Tank}: \mathit{GetPosition}(\mathtt{now}))$
$\mathtt{Del:}$ $\mathbf{in}(\mathtt{From}, \mathrm{Tank}: \mathit{GetPosition}(\mathtt{now}))$
$\mathtt{Add:}$ $\mathbf{in}(\mathtt{To}, \mathrm{Tank}: \mathit{GetPosition}(\mathtt{now}+1))$

Route Agent:

$\mathit{PlanRoute}(\mathtt{Map}, \mathtt{SourcePoint}, \mathtt{DestinationPoint}, \mathtt{VehicleType})$
$\mathtt{Pre:}$ $\mathtt{SourcePoint} \neq \mathtt{DestinationPoint}$
$\mathtt{Del:}$ $\{\}$
$\mathtt{Add:}$ $\mathbf{in}(\mathbf{true}, \mathrm{Route}: \mathit{UseMap}(\mathtt{Map}, \mathtt{now}))$ &
$\mathbf{in}(\mathtt{Plan}, \mathrm{Route}: \mathit{GetPlan}(\mathtt{SourcePoint}, \mathtt{DestinationPoint}, \mathtt{VehicleType}, \mathtt{now}))$

Coordination Agent:

$\mathit{Attack}(\texttt{SetOfAgentIds},\texttt{EnemyId})$
Pre: $\texttt{SetOfAgentIds} \neq \{\}$
Del: $\{\}$
Add: $\mathbf{in}(\texttt{AP}, \text{Coord}: \mathit{CoordinatedAttack}(\texttt{SetOfAgentIds}, \texttt{EnemyId}, \texttt{now}))$

3.8 Example (Message Box)
Throughout this paper, we will assume that each agent's associated software code includes a special type called `msgbox` *(short for message box). The message box is a buffer that may be filled (when it sends a message) or flushed (when it reads the message) by the agent. In addition, we assume the existence of an operating-systems level messaging protocol (e.g. SOCKETS or TCP/IP (Wilder 1993)) that can fill in (with incoming messages) or flush (when a message is physically sent off) this buffer.*

We will assume that the agent has the following functions that are integral in managing this message box. Note that over the years, we expect a wide variety of messaging languages to be developed (examples of such messaging languages include KQML (Labrou and Finin 1997) at a high level, and remote procedure calls at a much lower level). In order to provide maximal flexibility, we will merely specify below the core *interface functions available on the* `msgbox` *type. Note that this set of functions may be augmented by the addition of other functions on an agent by agent basis.*

- *SendMessage*(<*source_agent*>, <*dest_gent*>, <*message*>): *This causes a quintuple* (`o`, "*src*", "*dest*", "*message*", "*time*") *to be placed in* `msgbox`. *The parameter* `o` *signifies an outgoing message. When a call of SendMessage*("*src*", "*dest*", "*message*") *is executed, the state of* `msgbox` *changes by the insertion of the above quintuple denoting the sending of a message from the source agent* src *to a given Destination agent* dest *involving the message body "message"; "time" denotes the time at which the message was sent.*

- *GetMessage*(<*src*>): *This causes a collection of quintuples*

  (`i`, "*src*", "*agent*", "*msg*", "*time*")

  to be read from `msgbox`. *The i signifies an incoming message. Note that all messages from the given source to the agent* agent *whose message box is being examined, are returned by this operation. "time" denotes the time at which the message was received.*

- *TimedGetMessage*(<*op*>, <*valid*>): *This causes the collection of all quintuples tup of the form tup* $=_{def}$ (`i`, <*src*>, <*agent*>, <*message*>, `time`) *to be read from* `msgbox`, *such that the comparison* `tup.time` *op valid is true, where op is required to be any of the standard comparison operators* $<$, $>$, $\leq$, $\geq$, *or* $=$.

Agents interact with the external world through the `msgbox` code—in particular, external agents may update agent a's `msgbox`, thus introducing new objects to agent a's state, and triggering state changes which are not triggered by agent a.

3.9 Definition (Action Base $\mathcal{AB}$)
An action base, $\mathcal{AB}$, *is any finite collection of actions.*

In (Eiter, Subrahmanian, and Pick 1998), three alternative definitions of concurrent execution of actions are given. For one of those definitions, determining concurrent executability is polynomial time, for another it is NP-complete, and for the third, it is co-NP-complete. These complexities reflect definitions that are increasingly epistemically satisfying. Rather than reinvent the wheel here, we will merely assume the existence of a predicate, `conc_ex` which takes four arguments—a set of ground action atoms, a precondition, an add-list, and a delete list. Intuitively, `conc_ex(Aset, Pre, Add, Del)` means that the set, `Aset` of actions is concurrently executable and the concurrent execution of the actions in `Aset` may be viewed as a single "composite" action with the specified precondition, $Pre(\alpha)$, $Add(\alpha)$ and $Del(\alpha)$. In the event that the `Aset` is not concurrently executable, $Pre(\alpha)$ is set to the special condition **false**, reflecting the fact that the "composite" action is un-executable.

3.3.2 Action Constraints $\mathcal{AC}$

An *action constraint* AC is an explicit statement saying that a given set of actions is not concurrently executable if certain conditions are met.

3.10 Definition (Action Constraints $\mathcal{AC}$)
An action constraint AC has the syntactic form:

$$\{\alpha_1(\vec{X}_1), \ldots, \alpha_k(\vec{X}_k)\} \hookleftarrow \chi \tag{1}$$

where $\alpha_1(\vec{X}_1), \ldots, \alpha_k(\vec{X}_k)$ are action names, and χ is a code call condition.

A set of action constraints is denoted by $\mathcal{AC}$.

The above constraint says that if condition χ is true, then the actions $\alpha_1(\vec{X}_1), \ldots, \alpha_k(\vec{X}_k)$ are not concurrently executable.

3.11 Example (Constraints for Fly and Attack)
Here are two simple constraints for the Fly and the Attack predicate:

$\{$*Fly_plane1*`(X1 ,Y1 ,A1 ,S1)`,*Fly_plane2*`(X2 ,Y2 ,A2 ,S2)`$\}$ $\hookleftarrow$ `Y1 =Y2`
$\{$*Attack*`(P)`$\}$ $\hookleftarrow$ **in**(`P`, Heli : *GetPosition*(`now`))

3.3.3 Agent Programs

In this section, we introduce the important concept of an agent program. Intuitively, agent programs specify what an agent is obliged to do, what an agent is permitted to do, and what an agent is forbidden from doing. Agent programs provide a mechanism to encode the intended behavior of an agent.

3.12 Definition (Action Status Atom)
Suppose $\alpha(\vec{t})$ is an action atom, where $\vec{t}$ is a vector of terms (variables or objects) matching the type schema of α. Then, the formulas $\mathbf{P}(\alpha(\vec{t}))$, $\mathbf{F}(\alpha(\vec{t}))$, $\mathbf{O}(\alpha(\vec{t}))$, $\mathbf{W}(\alpha(\vec{t}))$, and $\mathbf{Do}(\alpha(\vec{t}))$ are action status atoms. *The set $\{\mathbf{P}, \mathbf{F}, \mathbf{O}, \mathbf{W}, \mathbf{Do}\}$ is called the action status set.*

We will often abuse notation and omit parentheses in action status atoms, writing $\mathbf{P}\alpha(\vec{t})$ instead of $\mathbf{P}(\alpha(\vec{t}))$, and so on. An action status atom has the following intuitive meaning (a more detailed description of the precise reading of these atoms will be provided later in Subsection 5.2):

- $\mathbf{P}\alpha$ means that the agent is permitted to take action α;
- $\mathbf{F}\alpha$ means that the agent is forbidden from taking α;
- $\mathbf{O}\alpha$ means that the agent is obliged to take action α;
- $\mathbf{W}\alpha$ means that obligation to take action α is waived; and,
- $\mathbf{Do}\alpha$ means that the agent does take action α.

Notice that the operators $\mathbf{P}, \mathbf{F}, \mathbf{O}$, and $\mathbf{W}$ have been extensively studied in the area of deontic logic (Meyer and Wieringa 1993; Åquist 1984). Moreover, the operator **Do** is in the spirit of the "praxiological" operator $\mathbf{E}_{\mathfrak{a}}A$ (Kanger 1972), which informally means that "agent $\mathfrak{a}$ sees to it that A is the case" (Meyer and Wieringa 1993, p.292).

3.13 Definition (Action Rule)
An action rule *(*rule*, for short) is a clause r of the form*

$$A \leftarrow L_1, \dots, L_n \tag{2}$$

where A is an action status atom, and each of $L_1, \dots, L_n$ is either an action status atom, or a code call atom, each of which may be preceded by a negation sign ($\neg$).

We require that every root variable which occurs in the head A of a rule r and every root- or path-variable occurring in a negative atom also occurs in some positive atom in the body (this is the well-known *safety* requirement on rules (Ullman 1989)).

A rule r is to be understood as being implicitly universally quantified over the variables in it. A rule is called *positive*, if no negation sign occurs in front of an action status atom in its body.

3.14 Definition (Agent Program)
An agent program *$\mathcal{P}$ is a finite collection of rules. An agent program $\mathcal{P}$ is* positive*, if all its rules are positive.*

4 Belief Language and Data Structures

In the following definition we introduce the most important notion, namely *belief atoms*. Belief atoms express the beliefs of one agent $\mathfrak{a}$ about what holds in another agent's, say $\mathfrak{b}$'s, state. They will be used later in Definition 4.13 on page 24 to define the notion of a *meta agent program*, which is central to this paper.

When an agent $\mathfrak{a}$ reasons about another agent $\mathfrak{b}$, it must have some beliefs about $\mathfrak{b}$'s underlying action base (*what actions can* $\mathfrak{b}$ *take?*), $\mathfrak{b}$'s action program (*how will* $\mathfrak{b}$ *reason?*) etc. These beliefs will be discussed later in more depth.

In this section, we will describe the belief language that is used by *IMPACT* agents. In particular, our definitions proceed as follows:

1. We first describe in Subsection 4.1 a hierarchy of belief languages of increasing complexity as we go "up" the hierarchy.
2. We then define in Subsection 4.2 an intermediate structure called a *basic belief table.* Intuitively, a basic belief table maintained by agent $\mathfrak{a}$ contains information about the beliefs $\mathfrak{a}$ has about the states of other agents, as well as $\mathfrak{a}$ itself. It also includes $\mathfrak{a}$'s belief about action status atoms that are adopted by other agents.
3. Each agent also has some beliefs about how other agents reason about beliefs. As the same syntactic language fragment can admit many different semantics, the agent maintains a *Belief Semantics Table*, describing its perceptions of the semantics used by other agents to reason about beliefs (Subsection 4.3).
4. We then extend in Subsection 4.4 the concept of a basic belief table to a *belief table.* Intuitively, a belief table is obtained by adding an extra column to the basic belief table—the reason for separating these two definitions is that the new column may refer to conditions on the columns of basic belief tables. Intuitively, belief tables contain statements of the form *If condition ϕ is true, then agent* $\mathfrak{a}$ *believes* ψ where ψ is a condition about some agent $\mathfrak{b}$'s state, or about the actions that agent $\mathfrak{b}$ might take.

Is important to note that assuming additional datatypes as part of our underlying software package has strong implications on the possible code calls as introduced in Definition 3.2 on page 8: the more datatypes we have, the more types of code calls can be formulated in our language. We will introduce in Definition 5.6 on page 28 a precise notion of the set of *extended code calls.*

4.1 Belief Language Hierarchy

We are now ready to start defining the beliefs that agent $\mathfrak{a}$ may hold about the code calls agent $\mathfrak{b}$ can perform. These code calls determine the code call conditions that may or may not hold in agent $\mathfrak{b}$'s state. Let us denote this by the belief atom

$$\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \chi)$$

which represents one of the beliefs of agent $\mathfrak{a}$ about what holds in the state of agent $\mathfrak{b}$. In that case, agent $\mathfrak{a}$ must have beliefs about agent $\mathfrak{b}$'s software package $\mathcal{S}^b$: the code call condition χ has to be contained in $\mathcal{S}^b$. We will collect all the beliefs that an agent $\mathfrak{a}$ has about another agent $\mathfrak{b}$ in a set $\Gamma^{\mathfrak{a}}(\mathfrak{b})$ (see Definition 5.5 on page 28).

From now on we will refer to code call conditions satisfying the latter property as *compatible code call conditions.* We will use the same term for action atoms: *compatible action atoms* of agent $\mathfrak{a}$ with respect to agent $\mathfrak{b}$, are those in the action base that $\mathfrak{a}$ believes agent $\mathfrak{b}$ holds. We also assume that the structure of such an action contained in $\mathfrak{b}$'s base (as believed by $\mathfrak{a}$) is defined in $\Gamma^{\mathfrak{a}}(\mathfrak{b})$. This means that the *schema*, the *set of preconditions*, the *add-list* and the *delete-list* are uniquely determined.

4.1 Definition (Belief Atom/Literal, $\mathcal{BAt}_1(\mathsf{a},\mathsf{b})$, $\mathcal{BLit}_1(\mathsf{a},\mathsf{A})$)
Let a,b *be agents in* A. *Then we define the set* $\mathcal{BAt}_1(\mathsf{a},\mathsf{b})$ *of* a*-belief atoms about* b *of level 1 as follows:*

1. *If* χ *is a compatible code call condition of* a *with respect to* b, *then* $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b},\chi)$ *is a* belief atom.
2. *For* $\mathbf{M} \in \{\mathbf{O},\mathbf{W},\mathbf{P},\mathbf{F},\mathbf{Do}\}$*: if* $\mathbf{M}\alpha(\vec{t})$ *is a compatible action atom of agent* a *with respect to* b, *then* $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b},\mathbf{M}\alpha(\vec{t}))$ *is a* belief atom.

If $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b},\chi)$ *is a belief atom, then* $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b},\chi)$ *and* $\neg\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b},\chi)$ *are called* belief literals of level 1, *the corresponding set is denoted by* $\mathcal{BLit}_1(\mathsf{a},\mathsf{b})$.

Let

$$\mathcal{BAt}_1(\mathsf{a},\mathsf{A}) =_{def} \bigcup_{b\in\mathsf{A}} \mathcal{BAt}_1(\mathsf{a},\mathsf{b}) \quad \textit{and} \quad \mathcal{BLit}_1(\mathsf{a},\mathsf{A}) =_{def} \bigcup_{b\in\mathsf{A}} \mathcal{BLit}_1(\mathsf{a},\mathsf{b})$$

be the set of all a*-belief atoms (resp. belief literals) relative to* A. *This reflects the idea that agent* a *can have beliefs about many agents in* A, *not just about a single one.*

Here are a couple of belief atoms from our RAMP example:

4.2 Example (Belief Atoms In RAMP)

- $\boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank1},\mathbf{in}(\mathtt{pos1},\mathsf{Tank1}:\mathit{GetPosition}(\mathtt{now})))$
  This belief atom says that the agent, Heli1 *believes that agent* Tank1*'s current state indicates that* Tank1*'s current position is* pos1.
- $\boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank1},\mathbf{F}\mathit{Attack}(\mathtt{now}+1))$
  This belief atom says that the agent, Heli1 *believes that agent* Tank1*'s current state indicates that it is forbidden for* Tank1 *to attack in the next step.*
- $\boldsymbol{\mathcal{B}}_{\mathsf{Heli3}}(\mathsf{Tank1},\mathbf{in}(\mathtt{pos2},\mathsf{Tank1}:\mathit{GetPosition}(\mathtt{now})))$
  This belief atom says that the agent, Heli3 *believes that agent* Tank1*'s current state indicates that* Tank1*'s current position is* pos2.
- $\boldsymbol{\mathcal{B}}_{\mathsf{Heli3}}(\mathsf{Tank1},\mathbf{O}\mathit{Drive}(\mathtt{pos1},\mathtt{pos2},35))$
  This belief atom says that the agent, Heli3 *believes that agent* Tank1*'s current state makes it obligatory for* Tank1 *to drive from location* pos1 *to* pos2 *at* 35 *miles per hour.*

It is important to note that these are *beliefs* held by agents Heli1 and Heli3, respectively. Any of them could be an incorrect belief.

Thus far, we have not allowed for nested beliefs. The language $\mathcal{BLit}_1(\mathsf{a},\mathsf{A})$ does not allow agent a to have beliefs of the form *"Agent* b *believes that agent* c*'s state contains code call condition* χ*"*, i.e. agent a cannot express beliefs it has about the beliefs of another agent.

The next definition introduces nested beliefs and also a general *belief language*. We introduce the following notation: for a given set X of formulae and a set $\mathcal{C}$ of connectives, let $\mathbf{Cl}_{\mathcal{C}}(X)$ be the closure of X under the connectives from $\mathcal{C}$.

4.3 Definition (Nested Beliefs $\mathcal{BL}it_i(\mathfrak{a},\mathfrak{b})$, Belief language $\mathcal{BL}_i^{\mathfrak{a}}$)
In the following let $\mathfrak{a},\mathfrak{b} \in \mathsf{A}$ *and* $\mathcal{C} =_{def} \{\&, \neg\}$. *In accordance with Definition 4.1 we denote by*

$$\mathcal{BA}t_0(\mathfrak{a},\mathfrak{b}) =_{def} \{\phi : \ \phi \text{ is a compatible code call condition or action atom}\}$$

the flat set of code call conditions or action atoms—no belief atoms are allowed. Furthermore, we define

$$\mathcal{BL}_0(\mathfrak{a},\mathfrak{b}) =_{def} \mathbf{Cl}_{\mathcal{C}}(\mathcal{BA}t_0(\mathfrak{a},\mathfrak{b}))$$
$$\mathcal{BL}_1(\mathfrak{a},\mathfrak{b}) =_{def} \mathbf{Cl}_{\mathcal{C}}(\mathcal{BA}t_1(\mathfrak{a},\mathfrak{b})),$$

i.e. the set of formulae obtained by closing the set $\mathcal{BA}t_0(\mathfrak{a},\mathfrak{b})$, *resp. the set* $\mathcal{BA}t_1(\mathfrak{a},\mathfrak{b})$, *under the connectives in* $\mathcal{C}$.

We call

$$\mathcal{BL}_0^{\mathfrak{a}} =_{def} \mathbf{Cl}_{\mathcal{C}}(\textstyle\bigcup_{\mathfrak{b}\in\mathsf{A}} \mathcal{BL}_0(\mathfrak{a},\mathfrak{b}))$$
$$\mathcal{BL}_1^{\mathfrak{a}} =_{def} \mathbf{Cl}_{\mathcal{C}}(\textstyle\bigcup_{\mathfrak{b}\in\mathsf{A}} \mathcal{BL}_1(\mathfrak{a},\mathfrak{b}))$$

the belief languages of agent $\mathfrak{a}$ of level 0, resp. of level 1. *To define nested belief literals we set for* $i > 1$

$$\mathcal{BA}t_i(\mathfrak{a},\mathfrak{b}) =_{def} \{\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b},\beta) : \ \beta \in \mathcal{BA}t_{i-1}(\mathfrak{b},\mathsf{A})\}.$$

and correspondingly $\mathcal{BL}it_i(\mathfrak{a},\mathfrak{b})$. $\mathcal{BL}it_i(\mathfrak{a},\mathsf{A}) =_{def} \bigcup_{\mathfrak{b}\in\mathsf{A}} \mathcal{BL}it_i(\mathfrak{a},\mathfrak{b})$ *is called the set of belief literals of depth* i. *We also define*

$$\mathcal{BA}t_\infty(\mathfrak{a},\mathsf{A}) =_{def} \bigcup_{i=0}^{\infty} \mathcal{BA}t_i(\mathfrak{a},\mathsf{A}), \quad \mathcal{BL}it_\infty(\mathfrak{a},\mathsf{A}) =_{def} \bigcup_{i=0}^{\infty} \mathcal{BL}it_i(\mathfrak{a},\mathsf{A}).$$

Now let

$$\mathcal{BL}_i(\mathfrak{a},\mathfrak{b}) =_{def} \mathbf{Cl}_{\mathcal{C}}(\mathcal{BA}t_i(\mathfrak{a},\mathfrak{b})),$$

and

$$\mathcal{BL}_i^{\mathfrak{a}} =_{def} \mathbf{Cl}_{\mathcal{C}}(\bigcup_{\mathfrak{b}\in\mathsf{A}} \mathcal{BL}_i(\mathfrak{a},\mathfrak{b})) \tag{3}$$

be the belief language of agent $\mathfrak{a}$ of level i. *Finally*

$$\mathcal{BL}_\infty^{\mathfrak{a}} =_{def} \mathbf{Cl}_{\mathcal{C}}(\bigcup_{i=0}^{\infty} \mathcal{BL}_i^{\mathfrak{a}}) \tag{4}$$

is the maximal belief language an agent $\mathfrak{a}$ *can have. Formulae in this language are also called* general belief formulae.

At first sight the last definition looks overly complicated. The reason is that every agent keeps track of only its *own* beliefs, and not of another agent's beliefs (we will see later in Lemma 4.9 that an agent may be able to simulate another agent's state). This means that a nested belief atom of the form $\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b},\boldsymbol{\mathcal{B}}_{\mathfrak{c}}(\mathfrak{d},\chi))$ does not make sense (because $\mathfrak{b} \neq \mathfrak{c}$) and is not allowed in the above definition.

Note also that the closure under $\mathcal{C}$ in Equation (3) allows us to use conjunctions with respect to different agents $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b},\chi) \wedge \boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{c},\chi')$. The closure in Equation (4) allows us to use in addition different nested levels of beliefs, like $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b},\chi) \wedge \boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{c},\boldsymbol{\mathcal{B}}_{\mathsf{c}}(\mathsf{d},\chi'))$. However, for most practical applications this additional freedom seems not to be necessary. We discuss this point again in Lemma 4.9.

Here are some belief formulae from the RAMP example (see Section 2 or Appendix B):

4.4 Example (Belief Formulae for RAMP)
The following are belief formulae from $\mathcal{BL}_0^{\mathsf{Heli1}}$, $\mathcal{BL}_1^{\mathsf{Tank1}}$ *and* $\mathcal{BL}_2^{\mathsf{Coord}}$.

- $\boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{pos1}, \mathsf{Tank1}: \mathit{GetPosition}(\mathtt{now})))$.
  This formula is in $\mathcal{BL}_0^{\mathsf{Heli1}}$. *It says that agent* Heli1 *believes that agent* Tank1*'s current state indicates that* Tank1*'s current position is* pos1.
- $\boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{pos1}, \mathsf{Tank1}: \mathit{GetPosition}(\mathtt{now}))))$.
  This formula is in $\mathcal{BL}_1^{\mathsf{Tank1}}$. *It says that agent* Tank1 *believes that agent* Heli1 *believes that agent* Tank1*'s current position is* pos1.
- $\boldsymbol{\mathcal{B}}_{\mathsf{Coord}}(\mathsf{Tank1}, \boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank2}, \mathbf{in}(\mathtt{pos2}, \mathsf{Tank2}: \mathit{GetPosition}(\mathtt{now})))))$.
  This formula is in $\mathcal{BL}_2^{\mathsf{Coord}}$. *It says that agent* Coord *believes that agent* Tank1 *believes that* Heli1 *believes that agent* Tank2*'s current position is* pos2.

However, the following formula does not belong to any language:

$$\boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{Pos1}, \mathsf{Tank}: \mathit{GetPosition}(\mathtt{now})))).$$

The reason for this is because in Heli1's state there can be no beliefs belonging to Tank1.

4.2 Basic Belief Table

We now describe how the agent keeps track of its beliefs about other agents and how these beliefs can be updated. The easiest way to structure a set of beliefs is to view it as a relational database structure. The notion of a basic belief table provides the starting point for defining how an agent maintains beliefs about other agents.

4.5 Definition (Basic Belief Table $\mathbf{BBT}^{\mathsf{a}}$)
Every agent a *has an associated* basic belief table $\mathbf{BBT}^{\mathsf{a}}$ *which is a set of pairs*

$$\langle \mathsf{h}, \phi \rangle$$

where $\mathsf{h} \in \mathsf{A}$, $\phi \in \mathcal{BL}_i^{\mathsf{h}}$.

For example, if the entry $\langle \mathsf{b}, \boldsymbol{\mathcal{B}}_{\mathsf{b}}(\mathsf{a},\chi) \rangle$ is in the table $\mathbf{BBT}^{\mathsf{a}}$, then this intuitively means that agent a believes that agent b has the code call condition χ among its own beliefs about agent a. Here $\phi \in \mathcal{BL}_1^{\mathsf{b}}$.

4.6 Example (Basic Belief Table for RAMP Agents)
We define suitable basic belief tables for agent Tank1 *(Table 1 on the following page) and* Heli1 *(Table 2 on the next page).*

Agent	Formula
$\mathsf{Heli1}$	$\mathbf{in}(\mathtt{posh1}, \mathsf{Heli1} : \mathit{GetPosition}(\mathtt{now}))$
$\mathsf{Heli2}$	$\boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{post1}, \mathsf{Tank1} : \mathit{GetPosition}(\mathtt{now})))$
$\mathsf{Tank2}$	$\boldsymbol{\mathcal{B}}_{\mathsf{Tank2}}(\mathsf{Heli2}, \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{pos3}, \mathsf{Tank1} : \mathit{GetPosition}(\mathtt{now}))))$

Table 1: A Basic Belief Table for agent $\mathsf{Tank1}$.

Agent	Formula
$\mathsf{Heli2}$	$\mathbf{in}(\mathtt{posh2}, \mathsf{Heli2} : \mathit{GetPosition}(\mathtt{now}))$
$\mathsf{Tank1}$	$\mathbf{in}(\mathtt{post1}, \mathsf{Tank1} : \mathit{GetPosition}(\mathtt{now}))$
$\mathsf{Tank1}$	$\boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \mathbf{in}(\mathtt{posh1}, \mathsf{Heli1} : \mathit{GetPosition}(\mathtt{now})))$
$\mathsf{Tank2}$	$\boldsymbol{\mathcal{B}}_{\mathsf{Tank2}}(\mathsf{Tank1}, \boldsymbol{\mathcal{B}}_{\mathsf{Tank2}}(\mathsf{Heli1}, \mathbf{in}(\mathtt{pos4}, \mathsf{Heli1} : \mathit{GetPosition}(\mathtt{now}))))$

Table 2: A Basic Belief Table for agent $\mathsf{Heli1}$.

These tables describe that $\mathsf{Tank1}$ *and* $\mathsf{Heli1}$ *work closely together and know their positions. Both believe that the other knows about both positions.* $\mathsf{Tank1}$ *also believes that* $\mathsf{Tank2}$ *believes that in* $\mathsf{Heli2}$*'s state,* $\mathsf{Tank1}$ *is in position* $\mathtt{pos3}$ *(which is actually wrong).*

$\mathsf{Heli1}$ *thinks that* $\mathsf{Tank2}$ *believes that* $\mathsf{Tank1}$ *believes that* $\mathsf{Heli1}$ *is in position* $\mathtt{pos4}$*, which is also wrong.*

What kind of operations should we support on belief tables? We distinguish between two different types:

1. For a given agent h, other than a, we may want to select all entries in the table having h as first argument.
2. For a given belief formula ϕ, we may be interested in all those entries, whose second argument "implies" (w.r.t. some underlying definition of entailment) the given formula ϕ.

The latter point motivates us to consider more general relations between belief formulae with respect to an epistemic background theory. This will extend the expressibility and usefulness of our overall framework. For example the background theory can contain certain epistemic axioms about beliefs or even certain inference rules and the relation between belief formulae can be the entailment relation with respect to the chosen background theory.

4.3 Belief Semantics Table

Agent a may associate different background theories with different agents: it may assume that agent h reasons according to semantics $\mathcal{B}Sem_{\mathsf{h}}^{\mathsf{a}}$ and assumes that agent h' adopts a stronger semantics $\mathcal{B}Sem_{\mathsf{h}'}^{\mathsf{a}}$. We will store the information in a separate relational data structure:

4.7 Definition (Belief Semantics Table $\mathbf{BSemT}^{\mathfrak{a}}$ of Agent $\mathfrak{a}$)
Every agent $\mathfrak{a}$ *has an associated* belief semantics table $\mathbf{BSemT}^{\mathfrak{a}}$ *which is a set of pairs*

$$\langle \mathfrak{h}, \mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}} \rangle$$

where $\mathfrak{h} \in \mathcal{A}$ *and* $\mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}}$ *is a belief semantics over* $\mathcal{BL}^{\mathfrak{h}}_{i}$ *and* $i \in \mathbb{N}$ *is fixed. I.e.* $\mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}}$ *determines an entailment relation*

$$\phi \models_{\mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}}} \psi$$

between belief formulae $\phi, \psi \in \mathcal{BL}^{\mathfrak{h}}_{i}$*. We also assume the existence of the following function (which constitutes an extended code call, see Definition 5.6 on page 28) over* $\mathbf{BSemT}^{\mathfrak{a}}$*:*

$$\mathbf{BSemT}^{\mathfrak{a}} : \mathrm{select}(\mathrm{agent}, =, \mathfrak{h}),$$

which selects all entries corresponding to a specific agent $\mathfrak{h} \in \mathcal{A}$*.*

4.8 Example (Belief Semantics Tables for RAMP Agents)
We shortly describe how suitable Belief Semantics Table for $\mathsf{Heli1}$ *and* $\mathsf{Tank1}$ *can look like. We have to define entailment relations* $\mathcal{B}Sem^{\mathsf{Tank1}}_{\mathsf{Tank2}}$*,* $\mathcal{B}Sem^{\mathsf{Tank1}}_{\mathsf{Heli1}}$*,* $\mathcal{B}Sem^{\mathsf{Tank1}}_{\mathsf{Heli2}}$*, and* $\mathcal{B}Sem^{\mathsf{Heli1}}_{\mathsf{Tank1}}$*,* $\mathcal{B}Sem^{\mathsf{Heli1}}_{\mathsf{Tank2}}$*,* $\mathcal{B}Sem^{\mathsf{Heli1}}_{\mathsf{Heli2}}$*. For simplicity we restrict these entailment relations to belief formulae of level at most* 1*, i.e.* $\mathcal{BL}^{\mathfrak{h}}_{1}$*.*

1. $\mathcal{B}Sem^{\mathsf{Heli1}}_{\mathsf{Tank1}}$*: The smallest entailment relation satisfying the schema*

$$\boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Tank1.1}, \chi) \rightarrow \chi.$$

This says that $\mathsf{Heli1}$ *believes that all beliefs of* $\mathsf{Tank1}$ *about* $\mathsf{Tank1.1}$ *are actually true:* $\mathsf{Tank1}$ *knows all about* $\mathsf{Tank1.1}$.

2. $\mathcal{B}Sem^{\mathsf{Heli2}}_{\mathsf{Tank2}}$*: The smallest entailment relation satisfying the schema*

$$\boldsymbol{\mathcal{B}}_{\mathsf{Tank2}}(\mathsf{Tank2.1}, \chi) \rightarrow \chi.$$

This describes that $\mathsf{Heli2}$ *believes that all beliefs of* $\mathsf{Tank2}$ *about* $\mathsf{Tank2.1}$ *are actually true:* $\mathsf{Tank2}$ *knows all about* $\mathsf{Tank2.1}$.

3. $\mathcal{B}Sem^{\mathsf{Tank1}}_{\mathsf{Heli1}}$*: The smallest entailment relation satisfying the schema*

$$\boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank1}, \chi) \rightarrow \chi.$$

This describes that $\mathsf{Tank1}$ *believes that if* $\mathsf{Heli1}$ *believes in* χ *for* $\mathsf{Tank1}$*, then this is true (*$\mathsf{Heli1}$ *knows all about* $\mathsf{Tank1}$*. A particular interesting instance of* χ *is* $\mathbf{in}(\mathtt{post1}, \mathsf{Tank1} : \mathit{GetPosition}(\mathtt{now}))$.

4. $\mathcal{B}Sem^{\mathsf{Tank1}}_{\mathsf{Heli1}}$*: The smallest entailment relation satisfying the schema*

$$\boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank2}, \chi) \wedge \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank2.1}, \chi) \rightarrow \chi.$$

This describes that $\mathsf{Tank1}$ *believes that if* $\mathsf{Heli1}$ *believes that* χ *is true both for* $\mathsf{Tank2}$ *and* $\mathsf{Tank2.1}$ *then this is actually true.*

The notion of a semantics used in the belief semantics table is very general: it can be an arbitrary relation on $\mathcal{BL}_i^{\mathsf{h}} \times \mathcal{BL}_i^{\mathsf{h}}$. We briefly illustrate *(1) which sort of semantics can be expressed* and *(2) how our framework can be suitably restricted for practical applications.*

The generality and flexibility of our framework can be seen by considering the following two simple axioms that can be built-in to a semantics:

$$\begin{array}{lccc} (1) & \boldsymbol{\mathcal{B}}_{\mathsf{h}_2}(\mathsf{h},\chi) & \Rightarrow & \boldsymbol{\mathcal{B}}_{\mathsf{h}_2}(\mathsf{h}',\chi) \\ (2) & \boldsymbol{\mathcal{B}}_{\mathsf{h}_2}(\mathsf{h},\chi) & \Rightarrow & \chi \end{array}$$

The first axiom refers to different agents h, h' while the second combines different *levels* of belief atoms: see Equations (3) and (4) and the discussion after Definition 4.3. In many applications, however, such axioms will not occur: $\mathsf{h} = \mathsf{h}'$ is fixed and the axioms operate on the same level i of belief formulae.

Thus it makes sense to consider simplified versions of semantics that are easy to implement and to handle. In fact, given the results of Eiter, Subrahmanian, and Pick (1998) and the various semantics Sem for agent programs, i.e. with no belief atoms, we now show how such a semantics Sem induces, in a natural way a semantics $\mathcal{B}Sem_{\mathsf{h}}^{\mathsf{a}}$ to be used in a belief semantics table. These semantics can be implemented and handled as built-ins. Entries in $\mathbf{BSemT}^{\mathsf{a}}$ can then look like

$$\begin{array}{c} \langle \mathsf{h}_1, \mathsf{Sem}_{feas} \rangle \\ \langle \mathsf{h}_2, \mathsf{Sem}_{rat} \rangle \\ \langle \mathsf{h}_3, \mathsf{Sem}_{reas} \rangle \end{array}$$

meaning that the agents h_i behave according to the indicated semantics, which are well understood for action programs without beliefs.

The idea is to use the semantics Sem of the action program $\mathcal{P}^{\mathsf{a}}(\mathsf{b})$ (that a believes b to have) for the evaluation of the belief formulae. However, this is a bit complicated by the fact that the computation of the semantics depends on various other parameters like the *state* and the *action* and *integrity constraints.*

Before stating the definition, we recall that a semantics Sem is a set of action status sets which depend on (1) an action program, (2) a set of action constraints, (3) a set of integrity constraints, and, finally, (4) the current state. The notation $\mathsf{Sem}_{\mathsf{h}}(\mathcal{P})$ only reflects the influence of (1) but (2)–(4) are equally important. For example, when the belief semantics table contains the entry $\langle \mathsf{h}_1, \chi \rangle$ where χ is a code call condition, χ is a belief of a about h_1's state. χ is therefore a condition on the state of h_1. In contrast, an entry $\langle \mathsf{h}_1, \mathbf{M}\alpha(\vec{t}) \rangle$, where $\mathbf{M}\alpha(\vec{t})$ is an action atom, is a belief of a on the actions that h_1 holds. Consequently action atoms can be seen as conditions on h_1's action program.

In the following lemma, we show how to define belief semantics defined on belief languages of level 0 and 1. But belief formulae contain *both* code call conditions and action atoms and those are, as just discussed, evaluated in different domains. Therefore for a formula ϕ which is a *conjunction* of code call conditions and action atoms, we let

CCC(ϕ) be the conjunction of all ccc's occuring in ϕ,
ACT(ϕ) be the conjunction of all action atom's occuring in ϕ.

4.9 Lemma (Sem for Agent Programs induces $\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{h}}$)
Let Sem *be the* reasonable, rational *or* preferential *semantics for agent programs (i.e. not containing beliefs). Suppose agent* $\mathfrak{a}$ *believes that agent* $\mathfrak{h}$ *reasons according to* Sem*. Let* $\mathcal{P}(\mathfrak{h})$ *be the agent program of* $\mathfrak{h}$ *and* $\mathcal{O}(\mathfrak{h})$, $\mathcal{AC}(\mathfrak{h})$ *and* $\mathcal{IC}(\mathfrak{h})$ *the state, action constraints and integrity constraints of* $\mathfrak{h}$*. Then there is a basic belief table* $\mathbf{BSemT}^{\mathfrak{a}}$ *and a belief semantics* $\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{h}}$ *induced by* Sem *such that*

- $\mathfrak{a}$ *believes in* $\mathfrak{h}$*'s state, and*
- $\mathfrak{a}$ *believes in all actions taken by* $\mathfrak{h}$ *with respect to* Sem *and* $\mathcal{P}(\mathfrak{h})$.

More generally: let $i \in \mathbb{N}$ *and suppose agent* $\mathfrak{a}$ *believes that agent* $\mathfrak{h}_1$ *believes that agent* $\mathfrak{h}_2$ *believes that* ... *believes that agent* $\mathfrak{h}_{i-1}$ *acts according to* $\mathcal{P}^{\mathfrak{a}}(\sigma)$ *(where* $\sigma =_{def} [\mathfrak{h}_1, \mathfrak{h}_2, \ldots, \mathfrak{h}_{i-1}]$*) and state* $\mathcal{O}(\sigma)$*. Then there is a basic belief table* $\mathbf{BSemT}^{\mathfrak{a}}$ *and a belief semantics* $\mathcal{BS}em^{\mathfrak{a}}_{\sigma}$ *induced by* Sem *on a suitably restricted subset of* $\mathcal{BL}^{\sigma}_0 \times \mathcal{BL}^{\sigma}_0$ *such that*

- $\mathfrak{a}$ *believes in* $\mathfrak{h}_{i-1}$*'s state, and*
- $\mathfrak{a}$ *believes in all actions taken by* $\mathfrak{h}_{i-1}$ *with respect to* Sem *and* $\mathcal{P}(\sigma)$.

Proof: We define a belief semantics $\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{h}}$ on $\mathcal{BL}^{\mathfrak{h}}_0 \times \mathcal{BL}^{\mathfrak{h}}_0$ with respect to a state $\mathcal{O}$, $\mathcal{AC}$, and $\mathcal{IC}$ as follows:

$$\phi \models_{\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{h}}} \psi \text{ by } \begin{cases} 1. & \mathrm{ACT}(\psi) \in \mathsf{Sem}_{\mathfrak{h}}(\mathcal{P}^{\mathfrak{a}}(\mathfrak{h}) \cup \{\mathrm{ACT}(\phi)\}) \text{ wrt. the state } \mathcal{O} \cup \mathrm{CCC}(\phi). \\ 2. & \mathcal{O} \cup \mathrm{CCC}(\phi) \models \mathrm{CCC}(\psi). \\ 3. & \mathcal{AC} \text{ are satisfied wrt. enlarged program.} \\ 4. & \mathcal{O} \cup \mathrm{CCC}(\phi) \models \mathcal{IC} \end{cases}$$

We now define a belief semantics $\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{h}}$ on $\mathcal{BL}^{\mathfrak{h}}_1 \times \mathcal{BL}^{\mathfrak{h}}_1$ with respect to a state $\mathcal{O}$, $\mathcal{AC}$, and $\mathcal{IC}$ as follows.

1. We restrict, as already discussed, to entailment relations that operate on the *same* level of beliefs. For level 0 we just defined such a relation.
2. For level 1 beliefs we also restrict to those that contain the same agent as first component: $\{\mathcal{B}_{\mathfrak{h}}(\mathfrak{c}, \phi) : \phi$ is a code call condition or an action atom$\}$.
3. For a belief formula ϕ of level 1 which has the form $\mathcal{B}_{\mathfrak{h}}(\mathfrak{c}, \phi_1) \wedge \cdots \wedge \mathcal{B}_{\mathfrak{h}}(\mathfrak{c}, \phi_n)$ we let
$$\mathrm{CCC}(\gamma) =_{def} \mathrm{CCC}(\phi_1) \wedge \cdots \wedge \mathrm{CCC}(\phi_n)$$
and
$$\mathrm{ACT}(\gamma) =_{def} \mathrm{ACT}(\phi_1) \wedge \cdots \wedge \mathrm{ACT}(\phi_n).$$
4. We define:
$$\phi \models_{\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{h}}} \psi \text{ by } \begin{cases} 1. & \mathrm{ACT}(\psi) \in \mathsf{Sem}_{\mathfrak{c}}(\mathcal{P}^{\mathfrak{a}}([\mathfrak{h}, \mathfrak{c}]) \cup \{\mathrm{ACT}(\phi)\}) \text{ wrt. } \mathcal{O} \cup \mathrm{CCC}(\phi). \\ 2. & \mathcal{O} \cup \mathrm{CCC}(\phi) \models \mathrm{CCC}(\psi). \\ 3. & \mathcal{AC} \text{ are satisfied wrt. enlarged program.} \\ 4. & \mathcal{O} \cup \mathrm{CCC}(\phi) \models \mathcal{IC} \end{cases}$$
The notation $\mathcal{P}^{\mathfrak{a}}([\mathfrak{h}, \mathfrak{c}])$ denotes the program that $\mathfrak{a}$ believes $\mathfrak{h}$ to believe about $\mathfrak{c}$. The sequences σ will be introduced in Definition 5.2 on page 26. ∎

4.4 Belief Tables

We are now ready to give the full definition of a belief table.

4.10 Definition (Belief Table $\mathbf{BT}^{\mathfrak{a}}$)
Every agent $\mathfrak{a}$ *has an associated* belief table $\mathbf{BT}^{\mathfrak{a}}$, *which consists of triples*

$$\langle \mathfrak{h}, \phi, \chi_{\mathcal{B}} \rangle$$

where $\mathfrak{h} \in \mathsf{A}$, $\phi \in \mathcal{BL}_i^{\mathfrak{h}}$ *and* $\chi_{\mathcal{B}} \in \mathcal{B}Cond^{\mathfrak{a}}(\mathfrak{h})$ *is a* belief condition of $\mathfrak{a}$ *to be defined below (see Definition 4.11 on the following page).*

We identify that part of $\mathbf{BT}^{\mathfrak{a}}$ *where the third entries are empty (or, equivalently,* true*) with the basic belief table introduced in Definition 4.5 on page 17. Thus, every belief table induces a (possibly empty) basic belief table.*

We also assume the existence of the following two functions over $\mathbf{BT}^{\mathfrak{a}}$*:*

$$\mathbf{BT}^{\mathfrak{a}} : \text{proj-select}(\text{agent}, =, \mathfrak{h})$$

which selects all entries of $\mathbf{BT}^{\mathfrak{a}}$ *of the form* $\langle \mathfrak{h}, \phi, \mathbf{true} \rangle$ *(i.e. corresponding to a specific agent* $\mathfrak{h} \in \mathsf{A}$ *and having the third entry empty) and projects them on the first two arguments, and*

$$\mathbf{BT}^{\mathfrak{a}} : \text{B-proj-select}(r, \mathfrak{h}, \phi)$$

for all $r \in \mathcal{R} =_{def} \{\Rightarrow, \Leftarrow, \Leftrightarrow\}$ *and for all belief formulae* $\phi \in \mathcal{BL}_{\infty}^{\mathfrak{h}}$*. This function selects all entries of* $\mathbf{BT}^{\mathfrak{a}}$ *of the form* $\langle \mathfrak{h}, \psi, \mathbf{true} \rangle$ *that contain a belief formula* ψ *which is in relation* r *to* ϕ *with respect to the semantics* $\mathcal{B}Sem_{\mathfrak{h}}^{\mathfrak{a}}$ *as specified in the belief semantics table* $\mathbf{BSemT}^{\mathfrak{a}}$ *and projects them on the first two arguments.*

For example, if we choose $\Rightarrow \in \mathcal{R}$ *as the relation* r *then*

$$(\psi \Rightarrow \phi) \in \mathcal{B}Sem_{\mathfrak{h}}^{\mathfrak{a}}$$

or, equivalently, $\models_{\mathcal{B}Sem_{\mathfrak{h}}^{\mathfrak{a}}} (\psi \Rightarrow \phi)$ *says* ϕ *is entailed by* ψ *relative to semantics* $\mathcal{B}Sem_{\mathfrak{h}}^{\mathfrak{a}}$*.*

We emphasize the fact that although the two introduced project-select functions are defined on the full belief table $\mathbf{BT}^{\mathfrak{a}}$, they can be thought of as operating on the induced basic belief table $\mathbf{BBT}^{\mathfrak{a}}$, which results from $\mathbf{BT}^{\mathfrak{a}}$ by projection on the first two arguments of those triples where the third entry is empty.

In the last definition we introduced the notion of a belief table but we did not yet specify the third entry in it, the *belief condition.* The role of such a belief condition is to extend the expressiveness of the basic belief table by restricting the applicability to particular states, namely those satisfying the belief condition. Intuitively, $\langle \mathfrak{b}, \phi, \chi_{\mathcal{B}} \rangle$ means that

> Agent $\mathfrak{a}$ believes that ϕ is true in agent $\mathfrak{b}$'s state, if the condition $\chi_{\mathcal{B}}$ holds.

Note that agent $\mathfrak{a}$ can only reason about his own state, which *contains* (through the belief table $\mathbf{BT}^{\mathfrak{a}}$ and the belief semantics table $\mathbf{BSemT}^{\mathfrak{a}}$) his beliefs as well as his underlying epistemic theory about other agent's states.

$\mathbf{BT}^{a}$ and $\mathbf{BSemT}^{a}$, taken together, *simulate* agent b's state as believed by agent a.

A belief condition $\chi_{\mathcal{B}}$ that occurs in an entry $\langle b, \phi, \chi_{\mathcal{B}} \rangle$ must therefore be evaluated in what agent a believes is agent b's state. This is important because the code call conditions must be compatible and therefore not only depend on agent a but also on agent b.

4.11 Definition (Belief Conditions $\mathcal{B}Cond^{a}(h)$)
The set $\mathcal{B}Cond^{a}(h)$ of belief conditions of agent a is defined inductively as follows:

1. *Every code call condition χ of agent a compatible with agent h is in $\mathcal{B}Cond^{a}(h)$.*
2. *If X is an entry in the basic belief table (or, equivalently the projection of an entry of the belief table $\mathbf{BT}^{a}$ on the first two arguments) or a variable over basic belief table tuples, then*

   $$\mathbf{in}(\mathtt{X}, \mathbf{BT}^{a} : \text{proj-select}(\text{agent}, =, h))$$

   is in $\mathcal{B}Cond^{a}(h)$.
3. *If X is an entry in the basic belief table or a variable over such entries, $r \in \mathcal{R}$, $\phi \in \mathcal{BL}_{i}^{a}$ and $h \in A$ then*

   $$\mathbf{in}(\mathtt{X}, \mathbf{BT}^{a} : \text{B-proj-select}(r, h, \phi))$$

   is in $\mathcal{B}Cond^{a}(h)$.
4. *If χ and χ' are in $\mathcal{B}Cond^{a}(h)$, then so are $\exists \mathtt{X} \chi$, and any conjunction $(\neg)\chi \& (\neg)\chi'$.*

As belief conditions corresponding to step 1. above will be checked in what agent a believes is agent b's state, we introduce the following notation:

- $h\_part(\chi) =_{def}$ *the subconjunction of χ consisting of all code call conditions* ***not involving*** $\mathbf{BT}^{a}$,
- $a\_part(\chi) =_{def}$ *the subconjunction of χ consisting of all code call conditions* ***that involve*** $\mathbf{BT}^{a}$.

Note that $h\_part(\chi)$ consists of conditions that have to be checked in what a believes is agent h's state, while $a\_part(\chi)$ refers to the belief tables of agent a.

Agent	**Formula**	**Condition**
Heli1	$\mathbf{in}(\mathtt{pos1}, \text{Heli1} : \mathit{GetPosition}(\mathtt{now}))$	**true**
Heli2	$\mathcal{B}_{\text{Heli2}}(\text{Tank1}, \mathbf{in}(\mathtt{P}, \text{Tank1} : \mathit{GetPosition}(\mathtt{now})))$	$\mathcal{B}cond_{1}^{\text{Tank1}}$
Tank2	$\mathcal{B}_{\text{Tank2}}(\text{Heli1}, \mathcal{B}_{\text{Heli1}}(\text{Tank1}, \mathbf{in}(\mathtt{P}, \text{Tank1} : \mathit{GetPosition}(\mathtt{now}))))$	$\mathcal{B}cond_{2}^{\text{Tank1}}$

Table 3: A Belief Table for agent Tank1.

4.12 Example (Belief Table for RAMP Agents Revisited)
We now extend our basic belief tables for agent $\mathsf{Tank1}$ *(Table 1 on page 18) and* $\mathsf{Heli1}$ *(Table 2 on page 18). Let* $\mathcal{B}cond_1^{\mathsf{Tank1}}$ *be* $\mathbf{in}(\mathtt{pos1}, \mathsf{Tank1}: GetPosition(\mathtt{now}))$ *and define* $\mathcal{B}cond_2^{\mathsf{Tank1}}$ *by*

$$\mathbf{in}(\langle \mathsf{Heli1}, \textit{belief atom}\rangle, \mathbf{BT}^{\mathsf{Tank1}} : \text{proj-select}(\text{agent}, =, \mathsf{Heli1})),$$

where

$$\textit{belief atom} =_{def} \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{pos1}, \mathsf{Tank1}: GetPosition(\mathtt{Now}))).$$

The first row in the table says that $\mathsf{Tank1}$ *believes that in* $\mathsf{Heli1}$*'s state the position for* $\mathsf{Heli1}$ *is* $\mathtt{pos1}$*, unconditionally.*

The second row in the belief table above, says that $\mathsf{Tank1}$ *believes that if* $\mathsf{Tank1}$*'s position is* $\mathtt{pos1}$*,* $\mathsf{Heli2}$ *believes that in* $\mathsf{Tank1}$*'s state the position of* $\mathsf{Tank1}$ *is* $\mathtt{pos1}$*.*

The third row in the belief table says that if $\mathsf{Tank1}$ *believes* $\mathsf{Heli1}$ *believes that* $\mathsf{Tank1}$*'s position is* $\mathtt{pos1}$*, then* $\mathsf{Tank2}$ *believes* $\mathsf{Heli1}$ *believes* $\mathsf{Tank1}$*'s position is* $\mathtt{pos1}$*.*

The table for $\mathsf{Heli1}$ *looks as shown in Table 4, where* $\mathcal{B}cond_1^{\mathsf{Heli1}}$ *stands for*

$$\mathbf{in}(\mathtt{pos1}, \mathsf{Heli1} : GetPosition(\mathtt{now}))$$

and $\mathcal{B}cond_2^{\mathsf{Tank1}}$ *is defined by*

$$\mathbf{in}(\langle \mathsf{Tank1}, \textit{belief atom}\rangle, \mathbf{BT}^{\mathbf{Heli1}} : \text{proj-select}(\text{agent}, =, \mathsf{Tank1})),$$

where

$$\textit{belief atom} =_{def}= \boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \mathbf{in}(\mathtt{pos1}, \mathsf{Heli1} : GetPosition(\mathtt{Now}))).$$

Agent	**Formula**	**Condition**
$\mathsf{Heli2}$	$\mathbf{in}(\mathtt{pos1}, \mathsf{Heli2}: GetPosition(\mathtt{now}))$	**true**
$\mathsf{Tank1}$	$\boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \mathbf{in}(\mathtt{p}, \mathsf{Heli1} : GetPosition(\mathtt{now})))$	$\mathcal{B}cond_1^{\mathsf{Heli1}}$
$\mathsf{Tank2}$	$\boldsymbol{\mathcal{B}}_{\mathsf{Tank2}}(\mathsf{Tank1}, \boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \mathbf{in}(\mathtt{p}, \mathsf{Heli1} : GetPosition(\mathtt{now}))))$	$\mathcal{B}cond_2^{\mathsf{Heli1}}$

Table 4: A Belief Table for agent $\mathsf{Heli1}$.

We are now in a position to formally express a meta agent program, i.e. a program which formalizes the actions and the circumstances under which an agent a will execute these actions based not only on its own state but also on its beliefs about other agent's states.

4.13 Definition (Meta Agent Program (map) $\boldsymbol{\mathcal{BP}}$)
A meta action rule, *(mar for short), for agent* a *is a clause* r *of the form*

$$A \leftarrow L_1, \ldots, L_n \tag{5}$$

where A *is an action status atom, and each of* $L_1, \ldots, L_n$ *is either a code call literal, an action literal or a belief literal from* $\mathcal{B}Lit_\infty(a, \mathsf{A})$*.*

A meta agent program, *(map for short), for agent* a *is a finite set* $\mathcal{BP}$ *of meta agent rules for* a*.*

4.14 Example (map's For RAMP-Agents)
Let Heli1 *'s meta agent program be as follows:*

$$
\begin{array}{lll}
\mathbf{P}\ \mathit{Attack}(\mathtt{P1}\ ,\mathtt{P2}) & \leftarrow & \mathcal{B}_{\mathsf{Heli1}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{P2}, \mathsf{Tank1}: \mathit{GetPosition}(\mathtt{Now})))\ , \\
& & \mathbf{P}\ \mathit{Fly}(\mathtt{P1}\ ,\mathtt{P3}\ ,\mathtt{A}\ ,\mathtt{S}), \\
& & \mathbf{P}\ \mathit{Attack}(\mathtt{P3}\ ,\mathtt{P2}).
\end{array}
$$

where $\mathit{Attack}(\mathtt{P1},\mathtt{P2})$ *is an action which means attack position* P2 *from position* P1*.* Heli1*'s program says* Heli1 *can attack position* P2 *from* P1 *if* Heli1 *believes* Tank1 *is in position* P2*,* Heli1 *can fly from* P1 *to another position* P3 *at altitude* A *and speed* S*, and* Heli1 *can attack position* P2 *from* P3*.*

Let Tank1*'s meta agent program be as follows:*

$$
\begin{array}{lll}
\mathbf{O}\ \mathit{Attack}(\mathtt{P1}\ ,\mathtt{P2}) & \leftarrow & \mathbf{O}\ \mathit{DriveRoute}([\mathtt{P0}\ ,\mathtt{P1}\ ,\mathtt{P2}\ ,\mathtt{P3}\ ],\mathtt{S}), \\
& & \mathcal{B}_{\mathsf{Tank1}}(\mathsf{Tank2}, \mathbf{in}(\mathtt{P2}, \mathsf{Tank2}: \mathit{GetPosition}(\mathtt{Now}))).
\end{array}
$$

If Tank1 *must drive through a point where it believes* Tank2 *is, it must attack* Tank2*.*

From now on we assume that the software package $\mathcal{S}^{\mathfrak{a}} = (\mathcal{T}^{\mathfrak{a}}_{\mathcal{S}}, \mathcal{F}^{\mathfrak{a}}_{\mathcal{S}})$ of each agent $\mathfrak{a}$ contains as distinguished data types

1. the belief table $\mathbf{BT}^{\mathfrak{a}}$, and
2. the belief semantics table $\mathbf{BSemT}^{\mathfrak{a}}$,

as well as the corresponding functions

$$\mathbf{BT}^{\mathfrak{a}} : \text{B-proj-select}(r, \mathfrak{h}, \phi) \text{ and } \mathbf{BSemT}^{\mathfrak{a}} : \text{select}(\text{agent}, =, \mathfrak{h}).$$

5 Semantics of Meta-Agent Programs

It remains to define the *semantics* of meta agent programs. As in the case of agent programs without any metaknowledge (we refer to the appendix where we provided the definitions to make this paper selfcontained), the basic notion upon which more sophisticated semantics will be based, is the notion of a *feasible status set* for a given meta agent program $\mathcal{BP}$. In order to do this we first have to introduce the notion of a *belief status set*, the counterpart of a status set for a meta agent program.

5.1 Definition (Belief Status Set $\mathcal{BS}$)
A belief status set $\mathcal{BS}$ *of agent* $\mathfrak{a}$*, also written* $\mathcal{BS}(\mathfrak{a})$*, is a set consisting of two kinds of elements:*

- *ground action status atoms over* $\mathcal{S}^{\mathfrak{a}}$ *and*
- *belief atoms from* $\mathcal{B}At_{\infty}(\mathfrak{a}, \mathsf{A})$ *of level greater or equal to 1.*

The reason that we do not allow belief atoms of level 0 is to avoid having code call conditions in our set. Such conditions are not implied by the underlying map (only action status atoms are allowed in the heads of rules). Moreover, in the agent programs without beliefs (which we want to extend) they are not allowed as well (see Definition A.1).

We note that such a set must be determined in accordance with

1. the map $\mathcal{BP}$ of agent $\mathfrak{a}$,
2. the current state $\mathcal{O}$ of $\mathfrak{a}$,
3. the underlying set of action and integrity constraints of $\mathfrak{a}$.

In contrast to agent programs without beliefs we now have to cope with all agents about which $\mathfrak{a}$ holds certain beliefs. Even if the map $\boldsymbol{\mathcal{BP}}$ does not contain nested beliefs (which are allowed), the belief table $\mathbf{BT}^{\mathfrak{a}}$ may and, by the belief semantics table $\mathbf{BSemT}^{\mathfrak{a}}$, such nested beliefs may imply (trigger) other beliefs. Thus we cannot restrict ourselves to belief atoms of level 1.

Any belief status set $\mathcal{BS}$ of agent $\mathfrak{a}$ induces, in a natural way, for any agent $\mathfrak{b} \in \mathrm{A}$, two sorts of sets: the *state* and the various *action status sets* that agent $\mathfrak{a}$ believes other agents $\mathfrak{b}$ to hold or those that $\mathfrak{a}$ believes other agents $\mathfrak{b}$ to hold about other agents $\mathfrak{c}$. To easily formalize the latter conditions, we introduce the notion of a sequence:

5.2 Definition (Sequence $\boldsymbol{\sigma}, [\boldsymbol{\rho}]$ of Agents)
A sequence σ of agents from A is defined inductively as follows:

1. *The empty sequence $[]$ is a sequence.*
2. *If $\mathfrak{a} \in \mathrm{A}$ and $[\rho]$ is a sequence, then $[\mathfrak{a}]$, $[-\mathfrak{a}]$, $[\mathfrak{a}, \rho]$, $[\rho, \mathfrak{a}]$, $[\mathfrak{a}, -\rho]$, $[-\rho, \mathfrak{a}]$ are sequences.*

We use both σ and $[\rho]$ to refer to an arbitrary sequence.

The negation signs in the last definition were introduced in order to distinguish between

$$\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \boldsymbol{\mathcal{B}}_{\mathfrak{b}}(\mathfrak{c}, \boldsymbol{\mathcal{B}}_{\mathfrak{c}}(\mathfrak{d}, \chi)))$$

and

$$\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \neg\boldsymbol{\mathcal{B}}_{\mathfrak{b}}(\mathfrak{c}, \boldsymbol{\mathcal{B}}_{\mathfrak{c}}(\mathfrak{d}, \chi))).$$

While the latter belief atom corresponds to the sequence $[-\mathfrak{b}, \mathfrak{c}, \mathfrak{d}]$, the former is described by $[\mathfrak{b}, \mathfrak{c}, \mathfrak{d}]$. The overall intuition of the formula $\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \boldsymbol{\mathcal{B}}_{\mathfrak{b}}(\mathfrak{c}, \boldsymbol{\mathcal{B}}_{\mathfrak{c}}(\mathfrak{d}, \chi)))$ is that if we keep agent $\mathfrak{a}$ in mind, then agent $\mathfrak{a}$ believes in a code call condition of type $[\mathfrak{b}, \mathfrak{c}, \mathfrak{d}]$, i.e. a ccc that $\mathfrak{b}$ believes that $\mathfrak{c}$ believes it holds in $\mathfrak{d}$'s state.

We also say sometimes "σ's state" and refer to the code call conditions that are true in what $\mathfrak{a}$ believes that $\mathfrak{b}$ believes ... where $[\mathfrak{a}, \mathfrak{b}, \dots] = \sigma$.

5.3 Definition (Induced Status Set $\Pi_{\mathsf{b}}^{\mathrm{action}}(\mathcal{BS})$ and State $\Pi_{\mathsf{b}}^{\mathrm{state}}(\mathcal{BS})$)
Let a, b *be agents and* $\mathcal{BP}$ *a* map *of* a. *Every belief status set* $\mathcal{BS}$ *of an agent* a *induces the following two sets describing* a*'s beliefs about* b*'s actions and* b*'s state*

$$\begin{array}{llcll}\Pi_{\mathsf{b}}^{\mathrm{action}}(\mathcal{BS}) & =_{def} & \{ \quad \mathbf{M}\alpha(\vec{t}) & : & \boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b}, \mathbf{M}\alpha(\vec{t})) \in \mathcal{BS},\ \textit{where } \mathbf{M} \in \{\mathbf{O}, \mathbf{W}, \mathbf{P}, \mathbf{F}, \mathbf{Do}\}\} \\ \Pi_{\mathsf{b}}^{\mathrm{state}}(\mathcal{BS}) & =_{def} & \{ \quad \chi & : & \boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b}, \chi) \in \mathcal{BS} \ \textit{ and } \chi \textit{ is a code call condition}\}\end{array}$$

Now assume that agent a *believes in* $\mathcal{BS}$. *Then* $\Pi_{\mathsf{b}}^{\mathrm{state}}(\mathcal{BS})$ *formalizes the state of agent* b *as believed by agent* a. *Similarly,* $\Pi_{\mathsf{b}}^{\mathrm{action}}(\mathcal{BS})$ *represents the action status set of agent* b *as believed by agent* a.

In the same way $\mathcal{BS}$ *induces for arbitrary sequences* σ *two sets*

$$\begin{array}{ll}\Pi_{\sigma}^{\mathrm{action}}(\mathcal{BS}) & \textit{describing } \mathsf{a}\textit{'s belief about actions corresponding to } \sigma \\ \Pi_{\sigma}^{\mathrm{state}}(\mathcal{BS}) & \textit{describing } \mathsf{a}\textit{'s belief about the state corresponding to } \sigma,\end{array}$$

depending on the depth of the belief atoms occuring in $\mathcal{BP}$.

It is important to note that for any sequence, σ of agents, $\Pi_{\sigma}^{\mathrm{action}}(\mathcal{BS})$ is a set of action status atoms. Likewise, $\Pi_{\sigma}^{\mathrm{state}}(\mathcal{BS})$ is a set of code call conditions that do *not* involve beliefs. For the empty sequence [], we identify $\Pi_{[]}^{\mathrm{action}}(\mathcal{BS})$ (resp. $\Pi_{[]}^{\mathrm{state}}(\mathcal{BS})$) with a's own action status set (resp. a's own state) as defined by the subset of $\mathcal{BS}$ not involving belief atoms.

5.4 Example (Belief Status Sets for RAMP-Agents)
We consider the map *of* Heli1 *given in Example 4.14*

$$\begin{array}{lll}\mathcal{BS}(\mathsf{Heli1}) =_{def} & \{ & \mathbf{P}\mathit{Fly}(\mathtt{PointA}, \mathtt{PointB}, 10000, 200), \mathbf{O}\mathit{Fly}(\mathtt{PointA}, \mathtt{PointB}, 10000, 200), \\ & & \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Heli2}, \mathbf{P}\mathit{Fly}(\mathtt{PointA}, \mathtt{PointB}, 10000, 200)), \\ & & \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Heli2}, \mathbf{in}(\mathtt{pos}, \mathsf{Heli1} : \mathit{GetPosition}(\mathtt{Now}))), \\ & & \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Heli2}, \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{pos}, \mathsf{Tank} : \mathit{GetPosition}(\mathtt{Now})))) \\ & & \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Heli2}, \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Tank1}, \mathbf{P}\mathit{Drive}(\mathtt{PointX}, \mathtt{PointY}, 40)))\}\end{array}$$

This belief status set is for Heli1 *and it says:*

1. *It is possible to fly from PointA to PointB at an altitude of 10000 feet and a speed of 200 knots.*

2. *It is obligatory to fly from PointA to PointB at an altitude of 10000 feet and a speed of 200 knots.*

3. Heli1 *believes that in* Heli2*'s state it is possible to fly from PointA to PointB at 10000 feet and 200 knots.*

4. Heli1 *believes that in* Heli2*'s state the position of* Heli2 *is* pos.

5. Heli1 *believes* Heli2 *believes that* Tank1*'s position is* pos.

6. Heli1 *believes* Heli2 *believes that in* Tank1*'s state it is possible to drive from* PointX *to* PointY *at 40 miles per hour.*

We then have:

$$\Pi^{\text{action}}_{\mathsf{Heli2}}(\mathcal{BS}(\mathsf{Heli1})) = \{\mathbf{P}\mathit{Fly}(\mathtt{PointA}, \mathtt{PointB}, 10000, 200)\}$$
$$\Pi^{\text{state}}_{\mathsf{Heli2}}(\mathcal{BS}(\mathsf{Heli1})) = \{\mathbf{in}(\mathtt{pos}, \mathsf{Heli1} : \mathit{GetPosition}(\mathtt{Now}))\}$$
$$\Pi^{\text{action}}_{[\mathsf{Heli2},\mathsf{Tank1}]}(\mathcal{BS}(\mathsf{Heli1})) = \{\mathbf{P}\mathit{Drive}(\mathtt{PointX}, \mathtt{PointY}, 40)\}$$
$$\Pi^{\text{state}}_{[\mathsf{Heli2},\mathsf{Tank1}]}(\mathcal{BS}(\mathsf{Heli1})) = \{\mathbf{in}(\mathtt{pos}, \mathsf{Tank1} : \mathit{GetPosition}(\mathtt{Now}))\}$$

These sets formalize the following:

- $\Pi^{\text{action}}_{\mathsf{Heli2}}(\mathcal{BS}(\mathsf{Heli1}))$ *describes* Tank1*'s beliefs about* Heli2*'s actions and it says that it is possible to fly from PointA to PointB at 10000 feet and 200 knots.*
- $\Pi^{\text{state}}_{\mathsf{Heli2}}(\mathcal{BS}(\mathsf{Heli1}))$ *describes* Tank1*'s beliefs about* Heli2*'s state and it says that its position is* pos.
- $\Pi^{\text{action}}_{[\mathsf{Heli2},\mathsf{Tank1}]}(\mathcal{BS}(\mathsf{Heli1}))$ *describes* Tank1*'s beliefs about* Heli2*'s beliefs about* Tank1*'s actions, and it says that it is possible to drive from PointX to PointY at 40 miles per hour.*
- $\Pi^{\text{state}}_{[\mathsf{Heli2},\mathsf{Tank1}]}(\mathcal{BS}(\mathsf{Heli1}))$ *describes* Tank1*'s beliefs about* Heli2*'s beliefs about* Tank1*'s state, and it says that its position is* pos.

Obviously for a to make a guess about agent b's behaviour, agent a not only needs a belief table and a belief semantics table, but a also needs to guess about b's action base, action program as well as the action and integrity constraints used by b. This is very much like having a guess about b's software package which we motivated and illustrated just before Definition 4.1 on page 15 (see the notion of *compatible* code call condition). For notational convenience and better readability we merge all these ingredients into a set $\Gamma^{\mathsf{a}}(\mathsf{b})$.

5.5 Definition ($\Gamma^{\mathsf{a}}(\mathsf{b})$, $\mathsf{Info}(\mathsf{a})$)
For agents $\mathsf{a}, \mathsf{b} \in A$, we denote by $\Gamma^{\mathsf{a}}(\mathsf{b})$ the following list of all beliefs that agent a holds about another agent b: the software package $\mathcal{S}^{\mathsf{a}}(\mathsf{b})$, the action base *$\mathcal{AB}^{\mathsf{a}}(\mathsf{b})$, the* action program *$\mathcal{P}^{\mathsf{a}}(\mathsf{b})$, the* integrity constraints *$\mathcal{IC}^{\mathsf{a}}(\mathsf{b})$ and the* action constraints *$\mathcal{AC}^{\mathsf{a}}(\mathsf{b})$. $\Gamma^{\mathsf{a}}(\mathsf{b})$ may also contain these objects for sequences $\sigma = [\mathsf{b}, \mathsf{c}]$ instead of b: we use therefore also the notation $\Gamma^{\mathsf{a}}([\mathsf{b}, \mathsf{c}])$. $\Gamma^{\mathsf{a}}(\sigma)$ represents a's beliefs about b's beliefs about c.*

In addition, given an agent a, we will often use the notation $\mathsf{Info}(\mathsf{a})$ to denote the software package $\mathcal{S}^{\mathsf{a}}$, the action base $\mathcal{AB}$, the action program $\mathcal{P}$, the integrity constraints $\mathcal{IC}$ and action constraints $\mathcal{AC}$ used by agent a. Thus we define $\mathsf{Info}(\mathsf{a}) =_{def} \Gamma^{[]}(\mathsf{a})$.

The set $\mathbf{\Gamma^{\mathsf{a}}(\mathsf{b})}$ is very important and therefore we introduce corresponding software code calls, thereby extending our original package $\mathcal{S}$.

5.6 Definition (Extended Code Calls, $\mathcal{S}^{\text{ext}}$)
Given an agent a, we will from now on distinguish (if it is not immediately clear from context) between basic *and* extended *code calls respectively code call conditions. The basic code calls refer to the package $\mathcal{S}$, while the latter refer to the extended software package which also contains*

1. *the following function of the belief table:*

(a) $\mathfrak{a}$: *belief_table*(), *which returns the full belief table of agent* $\mathfrak{a}$, *as a set of triples* $\langle \mathfrak{h}, \phi, \chi_{\mathcal{B}} \rangle$,

2. *the following functions of the belief semantics table:*

   (b) $\mathfrak{a}$: *belief_sem_table*(), *which returns the full belief semantics table, as a set of pairs* $\langle \mathfrak{h}, \mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}} \rangle$,

   (c) $\mathfrak{a}$: *bel_semantics*$(\mathfrak{h}, \phi, \psi)$, *which returns* **true** *when* $\phi \models_{\mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}}} \psi$ *and* **false** *otherwise.*

3. *the following functions, which implement for every sequence* σ *the beliefs of agent* $\mathfrak{a}$ *about* σ *as described in* $\mathbf{\Gamma^{\mathfrak{a}}(\boldsymbol{\sigma})}$:

   (d) $\mathfrak{a}$: *software_package*(σ), *which returns the set* $\mathcal{S}^{\mathfrak{a}}(\sigma)$,

   (e) $\mathfrak{a}$: *action_base*(σ), *which returns the set* $\mathcal{AB}^{\mathfrak{a}}(\sigma)$,

   (f) $\mathfrak{a}$: *action_program*(σ), *which returns the set* $\mathcal{P}^{\mathfrak{a}}(\sigma)$,

   (g) $\mathfrak{a}$: *integrity_constraints*(σ), *which returns the set* $\mathcal{IC}^{\mathfrak{a}}(\sigma)$

   (h) $\mathfrak{a}$: *action_constraints*(σ), *which returns the set* $\mathcal{AC}^{\mathfrak{a}}(\sigma)$,

4. *the following function which simulates the state of another agent* $\mathfrak{b}$ *or a sequence* σ,

   (i) $\mathfrak{a}$: *bel_ccc_act*(σ), *which returns all the code call conditions and action status atoms that* $\mathfrak{a}$ *believes are true in* σ*'s state. We write these objects in the form* "$\mathbf{in}(,)$" *(resp.* "$\boldsymbol{M}\alpha$" *for action status atoms) in order to distinguish them from those that have to be checked in* $\mathfrak{a}$*'s state.*

We also write $\mathcal{S}^{\text{ext}}$ *for this extended software package and distinguish it from the original* $\mathcal{S}$ *from which we started.*

5.1 Feasible Belief Status Sets

Consider now an agent $\mathfrak{a}$ with associated structures, $\mathsf{Info}(\mathfrak{a})$. Suppose $\mathcal{B}S$ is an arbitrary status set. We would like to first identify the conditions that determine whether it "makes sense" for agent $\mathfrak{a}$ to hold the set of beliefs prescribed by $\mathcal{B}S$. In particular, agent $\mathfrak{a}$ must use some epistemically well justified criteria to hold a set, $\mathcal{B}S$, of beliefs. In this section, we introduce the concept of a *feasible belief status set*. Intuitively, $\mathcal{B}S$ is feasible *if and only if* it satisfies two types of conditions—conditions on the agent $\mathfrak{a}$, and conditions on the beliefs of agent $\mathfrak{a}$ about other agents $\mathfrak{b}$ or sequences σ.

Conditions on agent $\mathfrak{a}$:

1. **Deontic and action consistency:** $\mathcal{B}S$ must not contain any inconsistencies. For example, $\mathcal{B}S$ may not contain action status atoms, $\mathbf{O}\alpha$ and $\mathbf{F}\alpha$ as these two action status atoms are mutually incompatible. Similarly, the set of actions taken by agent $\mathfrak{a}$ must not violate any action constraints, i.e. if $\text{Todo} = \{\alpha | \mathbf{Do}\alpha \in \mathcal{B}S\}$, then for each ground instance of an action constraint of the $\text{ActSet} \hookleftarrow \chi$, either χ is false in the current agent state, or $\text{ActSet} \not\subseteq \text{Todo}$.

2. **Deontic and action closure:** This condition says that $\mathcal{BS}$ must be closed under the deontic operations. For example, if $\mathbf{O}\alpha \in \mathcal{BS}$, then $\mathbf{P}\alpha \in \mathcal{BS}$, and so on.
3. **Closure under rules of $\mathcal{BP}$:** Furthermore, if we have a rule in $\mathcal{BP}$ having a ground instance whose body's code-call conditions are all true in the current agent state, and whose action status atoms and belief literals are true in $\mathcal{BS}$, then the head of that (ground) rule must be in $\mathcal{BS}$.
4. **State consistency:** Suppose we concurrently execute all actions in the set Todo. Then the new state that results must be consistent with the integrity constraints associated with agent $\mathfrak{a}$.

Conditions on beliefs of agent $\mathfrak{a}$ about other agents $\mathfrak{b}$:

5. **Local feasibility:** This condition requires that for any agent $\mathfrak{b}$, every induced status set $\Pi^{\mathrm{action}}_{\mathfrak{b}}(\mathcal{BS})$ is feasible (in the original sense) with respect to the induced state $\Pi^{\mathrm{state}}_{\mathfrak{b}}(\mathcal{BS})$ and action program $\mathcal{P}^{\mathfrak{a}}(\mathfrak{b})$. Furthermore a similar condition must hold for any sequence σ instead of just $\mathfrak{b}$.
6. **Compatibility with $\mathbf{BT}^{\mathfrak{a}}$:** We have to ensure that (1) all belief atoms of the basic belief table are contained in $\mathcal{BS}$ and that (2) whenever a belief condition is true, then the corresponding belief formula is true in $\mathcal{BS}$.
7. **Compatibility with $\mathbf{BSemT}^{\mathfrak{a}}$:** If $\langle \mathfrak{b}, \mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{b}} \rangle$ is an entry in $\mathbf{BSemT}^{\mathfrak{a}}$, we have to ensure that $\mathfrak{b}$'s induced state is closed under the semantics $\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{b}}$.

We are now ready to formalize the above 7 basic conditions through a sequence of definitions.

5.7 Definition (Deontic/Action Consistency)
A belief status set $\mathcal{BS}$ held by agent $\mathfrak{a}$ is said to be deontically consistent, *if, by definition, it satisfies the following rules for any ground action α and any sequence σ of agents (including the empty sequence):*

1. *If $\mathbf{O}\alpha \in \Pi^{\mathrm{action}}_{\sigma}(\mathcal{BS})$, then $\mathbf{W}\alpha \notin \Pi^{\mathrm{action}}_{\sigma}(\mathcal{BS})$.*
2. *If $\mathbf{P}\alpha \in \Pi^{\mathrm{action}}_{\sigma}(\mathcal{BS})$, then $\mathbf{F}\alpha \notin \Pi^{\mathrm{action}}_{\sigma}(\mathcal{BS})$.*
3. *If $\mathbf{P}\alpha \in \Pi^{\mathrm{action}}_{\sigma}(\mathcal{BS})$, then $\Pi^{\mathrm{state}}_{\sigma}(\mathcal{BS}) \models Pre(\alpha)$ (i.e. α is executable in $\Pi^{\mathrm{state}}_{\sigma}(\mathcal{BS})$).*

A belief status set $\mathcal{BS}$ is called action consistent, *if and only if for every ground action instance,* ActSet $\hookleftarrow \chi$*, of an action constraint in $\mathcal{AC}$, either χ is false in state $\mathcal{O}$ or $\mathcal{BS} \cap \{\mathbf{Do}\alpha \mid \alpha \in \mathrm{ActSet}\} = \emptyset$.*

Intuitively, the requirement of deontic consistency ensures that belief sets are internally consistent and do not have conflicts about whether an action should or should not be taken by agent $\mathfrak{a}$. Action consistency ensures that the agent cannot violate action constraints.

At this point, the reader may wonder why we need to ensure that deontic/action consistency requirements also apply to *sequences* of agents rather than to just agent $\mathfrak{a}$ by itself. The reason is that if we replaced all occurrences of σ in the preceding definition by the empty sequence [], i.e. we just look at $\mathfrak{a}$'s own action status set, then we may still encounter *deontic inconsistencies nested within beliefs.* For example, agent $\mathfrak{a}$'s belief set could contain both $\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{O}\alpha)$ and $\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{F}\alpha)$. In this case, agent $\mathfrak{a}$ believes that action α is both forbidden and obligatory for agent $\mathfrak{b}$—a state of affairs that is clearly inconsistent. It is to rule out such scenarios that we have defined deontic and action consistency as above,

5.8 Lemma (Deontic Closure)
Suppose $\mathcal{BS}$ is a belief status set held by agent $\mathfrak{a}$. The deontic closure *of $\mathcal{BS}$, denoted* $\mathbf{D\text{-}Cl}(\mathcal{BS})$, *is the minimal extension of $\mathcal{BS}$ by new belief atoms, such that the following condition holds:*

$$\textit{if } \mathbf{O}\alpha \in \Pi_{\sigma}^{\text{action}}(\mathcal{BS}) \textit{ then } \mathbf{P}\alpha \in \Pi_{\sigma}^{\text{action}}(\mathcal{BS}),$$

where α is any ground action and σ is any sequence of agents. We say that $\mathcal{BS}$ is deontically closed *if, by definition, $\mathcal{BS} = \mathbf{D\text{-}Cl}(\mathcal{BS})$.*

Again, this requirement forces an agent $\mathfrak{a}$'s belief status set to be closed—if agent $\mathfrak{a}$ believes that α is obligatory for agent $\mathfrak{b}$ to perform then it must also believe that agent $\mathfrak{b}$ is permitted to perform action α.

Proof: We have to show that such a minimal extension of $\mathcal{BS}$ exists. Let us define a sequence $\mathcal{BS}_i$, where $\mathcal{BS}_0 =_{def} \mathcal{BS}$ and

$$\mathcal{BS}_{i+1} =_{def} \mathcal{BS}_i \cup \{\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{P}\alpha) \mid \boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{O}\alpha) \in \mathcal{BS}_i\}.$$

Obviously, $\bigcup \mathcal{BS}_0^\infty$ is a minimal extension of $\mathcal{BS}$ as required in the statement. The general case for arbitrary sequences σ instead of just $[\mathfrak{b}\ ]$ is analogous. ∎

5.9 Lemma (Action Closure)
Suppose $\mathcal{BS}$ is a belief status set held by agent $\mathfrak{a}$. The action closure *of $\mathcal{BS}$, denoted* $\mathbf{A\text{-}Cl}(\mathcal{BS})$, *is the minimal extension of $\mathcal{BS}$ by new belief atoms, such that the following conditions hold:*

1. *if* $\mathbf{O}\alpha \in \Pi_{\sigma}^{\text{action}}(\mathcal{BS})$, *then* $\mathbf{Do}\alpha \in \Pi_{\sigma}^{\text{action}}(\mathcal{BS})$,
2. *if* $\mathbf{Do}\alpha \in \Pi_{\sigma}^{\text{action}}(\mathcal{BS})$, *then* $\mathbf{P}\alpha \in \Pi_{\sigma}^{\text{action}}(\mathcal{BS})$,

where α is any ground action and σ is any sequence of agents. We say that a status $\mathcal{BS}$ is action-closed, if $\mathcal{BS} = \mathbf{A\text{-}Cl}(\mathcal{BS})$ holds.

Intuitively, this lemma says that for all ground actions α, if agent $\mathfrak{a}$ believes that action α is obligatory for agent $\mathfrak{b}$, then agent $\mathfrak{a}$ must believe that agent $\mathfrak{b}$ will do it.

Proof: We have to show that such a minimal extension of $\mathcal{BS}$ exists and follow the proof of the last lemma. We define a sequence $\mathcal{BS}_i$, where $\mathcal{BS}_0 =_{def} \mathcal{BS}$ and

$$\mathcal{BS}_{i+1} =_{def} \mathcal{BS}_i \cup \{\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{P}\alpha) \mid \boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{Do}\alpha) \in \mathcal{BS}_i\} \cup \{\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{Do}\alpha) \mid \boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{O}\alpha) \in \mathcal{BS}_i\}.$$

Obviously, $\bigcup \mathcal{BS}_0^\infty$ is a minimal extension of $\mathcal{BS}$ as required in the statement. As in the previous lemma, the general case for arbitrary sequences σ instead of just $[\mathfrak{b}\ ]$ is analogous. ∎

We are now ready to start defining the notion of closure of a belief status set, $\mathcal{BS}$, under the rules of a map, $\mathcal{BP}$. First, we define an operator, $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS})$ that takes as input, a belief status set, $\mathcal{BS}$, and produces as output, another belief status set, obtained by applying the rules in $\mathcal{BP}$ with respect to the state $\mathcal{O}$ once.

5.10 Definition (Operator $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS})$)
Suppose $\mathcal{BP}$ is a map, and $\mathcal{O}$ is an agent state. Then, $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS})$ is defined to be the set of all ground action status atoms A such that there exists a rule in $\mathcal{BP}$ having a ground instance of the form $r : A \leftarrow L_1, \ldots , L_n$, which we denote by

$$A \leftarrow B^{+}_{cc}(r) \cup B^{-}_{cc}(r) \cup B^{+}_{other}(r) \cup B^{-}_{other}(r)$$

(in order to distinguish between positive/negative occurrences of code call atoms and non-code call atoms, i.e. action status literals and belief literals) such that:

1. *$B^{+}_{other}(r) \subseteq \mathcal{BS}$ and $\neg.B^{-}_{other}(r) \cap \mathcal{BS} = \emptyset$, and*
2. *every code call $\chi \in B^{+}_{cc}(r)$ succeeds in $\mathcal{O}$, and*
3. *every code call $\chi \in \neg.B^{-}_{cc}(r)$ does not succeed in $\mathcal{O}$, and*
4. *for every atom $\mathsf{Op}(\alpha) \in B^{+}(r) \cup \{A\}$ such that $\mathsf{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$, the action α is executable in state $\mathcal{O}$.*

Intuitively, the operator $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS})$ closes $\mathcal{BS}$ by applying all rules of the map $\mathcal{BP}$ once. The following example shows how this operator works, using our familiar RAMP example.

5.11 Example ($\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS})$ for RAMP)
We continue with Example 4.14 on page 25 and consider the following belief status set for Heli1*:*

$$\begin{aligned}\mathcal{BS}_1(\mathsf{Heli1}) =_{def} \{ \; & \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{B}, \mathsf{Tank1} : \mathit{GetPosition}(\mathtt{Now}))),\\ & \mathbf{P}\mathit{Fly}(\mathtt{A}, \mathtt{C}, 5000, 100), \mathbf{P}\mathit{Attack}(\mathtt{C}, \mathtt{B})\}\end{aligned}$$

Then $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS}_1(\mathsf{Heli1})) = \{\mathbf{P}\mathit{Attack}(\mathtt{A}, \mathtt{B})\}$.

Note that no belief atoms are present, because the definition of **App** *only specifies program rule heads and we cannot have belief atoms in rule heads. Also, the atoms $\mathbf{P}\mathit{Fly}(\mathtt{A}, \mathtt{C}, 5000, 100)$ and $\mathbf{P}\mathit{Attack}(\mathtt{C}, \mathtt{B})$ were not preserved because there are no rules to support them.*

5.12 Definition (Program Closure)
A belief status set, $\mathcal{BS}$, is said to be closed *with respect to a map, $\mathcal{BP}$, and an agent state, $\mathcal{O}$, if, by definition, $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS}) = \{\mathbf{M}\alpha \mid \mathbf{M}\alpha \in \mathcal{BS}$ where $\mathbf{M} \in \{\mathbf{O}, \mathbf{W}, \mathbf{P}, \mathbf{F}, \mathbf{Do}\}\}$.*

Intuitively, this definition says that when we restrict $\mathcal{BS}$ to the action status atoms associated with agent a, then the set of action status atoms that the map, $\mathcal{BP}$, makes true in the current state, is equal to the set of action status atoms already true in $\mathcal{BS}$. The following example builds upon the previous one, and explains why certain belief status sets, $\mathcal{BS}$, satisfy the program closure condition, while others do not.

In the previous example the belief status set $\mathcal{BS}_1(\mathsf{Heli1})$ does not satisfy the program closure property because $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS}_1(\mathsf{Heli1}))$ is not equal to

$$\{M\alpha :\ M\alpha \in \mathcal{BS}\} = \{\mathbf{P}\mathit{Fly}(\mathtt{A},\mathtt{C},5000,100), \mathbf{P}\mathit{Attack}(\mathtt{C},\mathtt{B})\}.$$

However, if we add to Heli1's program the rules:

$$\begin{array}{ll} \mathbf{P}\ \mathit{Fly}(\mathtt{A}\ ,\ \mathtt{C}\ ,\ 5000\ ,100) & \leftarrow \\ \mathbf{P}\ \mathit{Attack}(\mathtt{C}\ ,\ \mathtt{B}) & \leftarrow \end{array}$$

the following belief status set $\mathcal{BS}_2(\mathsf{Heli1})$ does satisfy the program closure rule:

$$\begin{array}{lll} \mathcal{BS}_2(\mathsf{Heli1}) =_{def} & \{ & \boldsymbol{\mathcal{B}}_{\mathsf{Heli1}}(\mathsf{Tank1}, \mathbf{in}(\mathtt{B}, \mathsf{Tank}\colon \mathit{GetPosition}(\mathtt{Now}))), \\ & & \mathbf{P}\mathit{Fly}(\mathtt{A},\mathtt{C},5000,100), \\ & & \mathbf{P}\mathit{Attack}(\mathtt{C},\mathtt{B}), \\ & & \mathbf{P}\mathit{Attack}(\mathtt{A},\mathtt{B})\ \}. \end{array}$$

Then

$$\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS}_2(\mathsf{Heli1})) = \{\mathbf{P}\mathit{Attack}(\mathtt{A},\mathtt{B}), \mathbf{P}\mathit{Attack}(\mathtt{C},\mathtt{B}), \mathbf{P}\mathit{Fly}(\mathtt{A},\mathtt{C},5000,100)\}.$$

At this point, we have completed describing the requirements on agent $\mathfrak{a}$ that must be true. In addition, we must specify conditions on the *beliefs* that agent $\mathfrak{a}$ holds about other agents, $\mathfrak{b}$. To some extent, this has already been done in the definitions of deontic and action consistency/closure. However, more coherent conditions need to be articulated. The first of these is the fact that the beliefs held by agent $\mathfrak{a}$ about another agent $\mathfrak{b}$ must be coherent. For instance, if $\mathfrak{a}$ believes that it is obligatory for agent $\mathfrak{b}$ to do action α, then $\mathfrak{a}$ must also believe that $\mathfrak{b}$ will do α. Other, similar conditions also apply. This condition may be expressed through the following definition.

5.13 Definition (Local Coherence)
A belief status set, $\mathcal{BS}$, held by agent $\mathfrak{a}$ is said to be locally coherent w.r.t. a sequence, σ of agents *if, by definition, the induced status set $\Pi_\sigma^{\mathrm{action}}(\mathcal{BS})$ is feasible in the sense of (Eiter, Subrahmanian, and Pick 1998) with respect to the induced state $\Pi_\sigma^{\mathrm{state}}(\mathcal{BS})$ and agent program $\mathcal{P}^{\mathfrak{a}}(\sigma)$.*

$\mathcal{BS}$ is said to be locally coherent *if, by definition, $\mathcal{BS}$ is coherent with respect to all sequences, σ, of agents.*

The above definition makes explicit reference to the definition of feasible status set, provided by (Eiter, Subrahmanian, and Pick 1998). It is important to note that $\Pi_\sigma^{\mathrm{action}}(\mathcal{BS})$ is a set of action status atoms, and that $\Pi_\sigma^{\mathrm{state}}(\mathcal{BS})$ involves no belief literals, and $\mathcal{P}^{\mathfrak{a}}(\sigma)$ is an *agent program* with no belief modalities, as defined earlier on in Definition 3.14 on page 13. Here are a few examples of what it means for a belief status set held by agent $\mathfrak{a}$ to be locally coherent.

Let

$$\begin{array}{lll} \mathcal{BS}(\mathsf{Heli2}) =_{def} & \{ & \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Heli1}, \mathbf{P}\mathit{Fly}(\mathtt{PointA},\mathtt{PointB},1000,100)), \\ & & \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Heli1}, \mathbf{in}(100, \mathsf{Heli1} : \mathit{GetAltitude}(\mathtt{now}))), \\ & & \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Heli1}, \mathbf{in}(1000, \mathsf{Heli1} : \mathit{GetAltitude}(\mathtt{now})))\ \} \end{array}$$

and let Heli1's program as believed by Heli2 (we denote it by $\mathcal{P}^{\mathsf{Heli2}}(\mathsf{Heli1})$) be:

$\mathbf{P}\ \mathit{Fly}(\mathtt{X}\ ,\mathtt{Y}\ ,\mathtt{A}\ ,\mathtt{S}) \leftarrow\ \ \mathbf{in}(\mathtt{S}, \mathsf{Heli1} : \mathit{GetSpeed}(\mathtt{now})),\ \ \mathbf{in}(\mathtt{A}, \mathsf{Heli1} : \mathit{GetAltitude}(\mathtt{now}))$.

The set $\mathcal{BS}(\mathsf{Heli2})$ is locally coherent w.r.t. the sequence $(\mathsf{Heli2})$. Notice that:

$$\Pi^{\text{action}}_{\mathsf{Heli1}}(\mathcal{BS}(\mathsf{Heli2})) =_{def} \{\mathbf{P}\mathit{Fly}(\mathtt{PointA}, \mathtt{PointB}, 1000, 100)\}$$

is feasible with respect to:

$$\Pi^{\text{state}}_{[\mathsf{Heli1}]}(\mathcal{BS}(\mathsf{Heli2})) = \{\mathbf{in}(100, \mathsf{Heli1} : \mathit{GetAltitude}(\mathtt{now})), \mathbf{in}(1000, \mathsf{Heli1} : \mathit{GetAltitude}(\mathtt{now}))\}.$$

Let

$$\begin{array}{lll} \mathcal{BS}(\mathsf{Tank1}) =_{def} & \{ & \boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli2}, \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Heli1}, \mathbf{P}\mathit{Fly}(\mathtt{PointA}, \mathtt{PointB}, 1000, 100))), \\ & & \boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli2}, \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Heli1}, \mathbf{in}(100, \mathsf{Heli1} : \mathit{GetSpeed}(\mathtt{now})))), \\ & & \boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli2}, \boldsymbol{\mathcal{B}}_{\mathsf{Heli2}}(\mathsf{Heli1}, \mathbf{in}(1000, \mathsf{Heli1} : \mathit{GetAltitude}(\mathtt{now})))) \\ & \} & \end{array}$$

and let the $\mathcal{P}^{\mathsf{Tank1}}([\mathsf{Heli2}, \mathsf{Heli1}])$ be the program $\mathsf{Tank1}$ believes $\mathsf{Heli2}$ believes $\mathsf{Heli1}$ has:

$\mathbf{P}\ \mathit{Fly}(\mathtt{X}\ ,\mathtt{Y}\ ,\mathtt{A}\ ,\mathtt{S}) \leftarrow\ \ \mathbf{in}(\mathtt{S}, \mathsf{Heli1} : \mathit{GetSpeed}(\mathtt{now})),\ \ \mathbf{in}(\mathtt{A}, \mathsf{Heli1} : \mathit{GetAltitude}(\mathtt{now}))$.

Then $\mathcal{BS}(\mathsf{Tank1})$ is locally coherent w.r.t. the sequence $[\mathsf{Heli2}, \mathsf{Heli1}]$. Just like in the previous example: $\Pi^{\text{action}}_{[\mathsf{Heli2},\mathsf{Heli1}]}(\mathcal{BS}(\mathsf{Tank1})) = \{\mathbf{P}\mathit{Fly}(\mathtt{PointA}, \mathtt{PointB}, 1000, 100)\}$ is feasible with respect to:

$$\Pi^{\text{state}}_{[\mathsf{Heli2},\mathsf{Heli1}]}(\mathcal{BS}(\mathsf{Tank1})) = \{\mathbf{in}(100, \mathsf{Heli1} : \mathit{GetSpeed}(\mathtt{now})), \mathbf{in}(1000, \mathsf{Heli1} : \mathit{GetAltitude}(\mathtt{now}))\}.$$

In addition to being locally coherent, for a belief status set to be considered feasible, we need to ensure that it does not ignore the contents of the belief table of agent a. This may be encoded through the following condition.

5.14 Definition (Compatibility with $\mathbf{BT}^{\mathsf{a}}$)
Suppose $\langle \mathsf{h}, \phi, \chi_{\mathcal{B}} \rangle$ is in $\mathbf{BT}^{\mathsf{a}}$. $\mathcal{BS}$ is said to be compatible *with $\langle \mathsf{h}, \phi, \chi_{\mathcal{B}} \rangle$ if, by definition, either*

1. *$\mathsf{h\_part}(\chi_{\mathcal{B}})$ is false w.r.t. $\Pi^{\text{state}}_{\mathsf{h}}(\mathcal{BS})$ or $\mathsf{a\_part}(\chi_{\mathcal{B}})$ is false w.r.t. a's state $\mathcal{O}_{\mathcal{S}}$.*
2. *ϕ is a code call condition or an action status atom and $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{h}, \phi) \in \mathcal{BS}$.*

$\mathcal{BS}$ is said to be compatible *with $\mathbf{BT}^{\mathsf{a}}$ if, by definition, it is compatible with all tuples in $\mathbf{BT}^{\mathsf{a}}$.*

Intuitively, this condition says that if a row in the belief table of agent a has a "true" condition, then agent a must hold the corresponding belief about the agent h in question. The following example illustrates this concept of compatibility.

5.15 Example (Compatibility)
We continue with Table 3 on page 23. We define

$$\mathcal{BS}(\mathsf{Tank1}) =_{def} \{\boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \mathbf{in}(\mathtt{pos1}, \mathsf{Heli1} : \mathit{GetPosition}(\mathtt{now})))\}.$$

belss($\mathsf{Tank1}$*) is compatible with* $\mathbf{BT}^{a}$.

However, the following belief set is not compatible with the give belief table:

$$\mathcal{BS}(\mathsf{Tank1}) = \{\mathbf{in}(\mathtt{pos1}, \mathsf{Tank1} : \mathit{GetPosition}(\mathtt{now}))\}.$$

This is because there is a true condition in the first row of the table but the belief set does not contain

$$\boldsymbol{\mathcal{B}}_{\mathsf{Tank1}}(\mathsf{Heli1}, \mathbf{in}(\mathtt{pos1}, \mathsf{Heli1} : \mathit{GetPosition}(\mathtt{now})))$$

according to the definition of compatibility.

The last condition in defining feasible belief status sets is that for any agent b, the beliefs agent a holds about agent b must be closed under the notion of entailment that agent a thinks agent b uses.

5.16 Definition (Compatibility with BSemT^{a})
Suppose $\langle \mathsf{b}, \mathcal{B}Sem^{\mathsf{a}}_{\mathsf{b}}\rangle$ *is an entry in* $\mathbf{BSemT}^{a}$*, and suppose* $\mathcal{BS}$ *is a belief status set. Let* $\mathcal{BS}[\mathsf{b}] = \{\chi \,|\, \boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b}, \chi) \in \mathcal{BS}\}$*.* $\mathcal{BS}$ *is said to be* compatible *with* $\langle \mathsf{b}, \mathcal{B}Sem^{\mathsf{a}}_{\mathsf{b}}\rangle$ *if, by definition,*

$$\{\chi' \mid \mathcal{BS}[\mathsf{b}] \models_{\mathcal{B}Sem^{\mathsf{a}}_{\mathsf{b}}} \chi'\} \quad \subseteq \quad \mathcal{BS}.$$

$\mathcal{BS}$ *is said to be* compatible *with* $\mathbf{BSemT}^{a}$ *iff* $\mathcal{BS}$ *is compatible with every entry in* $\mathbf{BSemT}^{a}$.

We are now ready to define feasible belief status sets.

5.17 Definition (Feasible Belief Status Set)
A belief status set, $\mathcal{BS}$ *held by agent* a*, is said to be* feasible *with respect to a meta-agent program,* $\mathcal{BP}$*, an agent state,* $\mathcal{O}$*, and a set* $\mathcal{IC}$ *of integrity constraints, and a set* $\mathcal{AC}$ *of action constraints if, by definition,* $\mathcal{BS}$ *satisfies our 7 conditions stated above (deontically and action consistent, deontically and action closed, closed under the* map $\mathcal{BP}$*'s rules, state consistent, locally coherent, compatible with* $\mathbf{BT}^{a}$*, and compatible with* $\mathbf{BSemT}^{a}$*).*

To George 1 *George, please fill in 4 examples here. Include two feasible status sets, and 2 sets that are not feasible for different reasons.*

5.2 Rational Belief Status Sets

The notion of a rational status set is a useful strengthening of feasible status sets. The idea is that all executed actions should be *grounded* or *justified* by the meta agent program. As a simple example, consider a feasible belief status set and add a $\mathbf{Do}\alpha$ atom for an action α that does not occur in any rule of the program or in the action and integrity constraints at all. It is immediate that this new set still is a feasible belief status set, although not a minimal one: there is no reason to believe in $\mathbf{Do}\alpha$. Rational sets rule out such non-minimal status sets:

5.18 Definition (Groundedness; Rational Status Set)
A belief status set $\mathcal{BS}$ which is locally coherent, compatible with $\mathbf{BT}^a$, *and compatible with* $\mathbf{BSemT}^a$ *is* grounded, *if there exists no belief status set $\mathcal{BS}'$ strictly contained in $\mathcal{BS}$ ($\mathcal{BS}' \subset \mathcal{BS}$) such that $\mathcal{BS}'$ satisfies the following 3 conditions of a feasible belief status set as given in Definition 5.17: deontically and action consistent, deontically and action closed, closed under the* map *$\mathcal{BP}$'s rules.*

A belief status set $\mathcal{BS}$ is a rational status set, *if $\mathcal{BS}$ is a feasible status set and $\mathcal{BS}$ is grounded.*

If we compare this last definition with the original definition of a belief status set, Definition 5.17, the reader will note that only the state consistency is not explicitly required while minimizing $\mathcal{BS}'$. In contrast, the locally coherence and the two compatibility conditions are required and do not guide the minimization process. If state consistency were *added* to the minimization policy, then an agent would be forced to execute actions in order to satisfy the integrity constraints. However, such actions may not be mentioned at all by the program, and thus it seems unreasonable to execute them. Of course, the state consistency is guaranteed, because we check groundedness only for *feasible* belief status sets.

5.3 Reasonable Belief Status Sets

As shown in (Eiter, Subrahmanian, and Pick 1998) for programs without beliefs, rational status sets allow the arbitrary contraposition of rules, which is often not intended. For example the program consisting of the simple rule

$$\mathbf{Do}(\alpha) \leftarrow \neg\mathbf{Do}(\beta)$$

has two rational status sets: $S_1 = \{\mathbf{Do}(\alpha), \mathbf{P}(\alpha)\}$, and $S_2 = \{\mathbf{Do}(\beta), \mathbf{P}(\beta)\}$. The second one seems less intuitive because there is no rule in the program to justify deriving $\mathbf{Do}(\beta)$.

This leads, in analogy to (Eiter, Subrahmanian, and Pick 1998) to the following notion:

5.19 Definition (Reasonable Status Set)
Let $\mathcal{BP}$ be an agent program, let $\mathcal{O}_{\mathcal{S}}$ be an agent state, and let $\mathcal{BS}$ be a belief status set.

1. *If $\mathcal{BP}$ is a positive meta agent program, then $\mathcal{BS}$ is a* reasonable belief status set *for $\mathcal{P}$ on $\mathcal{O}_{\mathcal{S}}$, if, by definition, $\mathcal{BS}$ is a rational belief status set for $\mathcal{P}$ on $\mathcal{O}_{\mathcal{S}}$.*
2. *The reduct of $\mathcal{BP}$ w.r.t. $\mathcal{BS}$ and $\mathcal{O}_{\mathcal{S}}$, denoted by $red^{\mathcal{BS}}(\mathcal{BP}, \mathcal{O}_{\mathcal{S}})$, is the program which is obtained from the ground instances of the rules in $\mathcal{BP}$ over $\mathcal{O}_{\mathcal{S}}$ as follows.*
   (a) *First, remove every rule r such that $B^-_{other}(r) \cap \mathcal{BS} \neq \emptyset$;*
   (b) *Remove all atoms in $B^-_{other}(r)$ from the remaining rules.*

   Then $\mathcal{BS}$ is a reasonable status set *for $\mathcal{BP}$ w.r.t. $\mathcal{O}_{\mathcal{S}}$, if it is a reasonable status set of the program $red^{\mathcal{BS}}(\mathcal{BP}, \mathcal{O}_{\mathcal{S}})$ with respect to $\mathcal{O}_{\mathcal{S}}$.*

Analogously to (Eiter, Subrahmanian, and Pick 1998), we have the following relation between reasonable and rational status sets:

5.20 Theorem (Reasonable Status Sets are Rational)
Every reasonable status set is also rational.

6 How to Implement Meta-Agent Programs?

Meta-Agent Programs significantly extend agent programs by allowing to reason about beliefs. But within the *IMPACT*-platform developed at the University of Maryland, agent programs have been already efficiently implemented and thus the question arises if we can take advantage of this work. In fact, as we will show in this section, this can be done by

1. *transforming meta agent programs into agent programs*, and
2. *taking advantage of extended code calls $\mathcal{S}^{ext}$ as introduced in Definition 5.6.*

The first step is a source-to-source transformation: the belief atoms in a meta agent program are replaced by suitable code calls to the new datastructures. We also note that the second step is indispensable, as every agent dealing with meta agent programs needs to deal with *Belief Tables*, *Belief Semantics Tables* and some functions operating on them.

Let us illustrate the transformation with the following simplified example. Recall that we already introduced *extended* code call conditions in Definition 5.6 on page 28: those also involve the new datatypes (belief- and belief semantics tables). Suppose the belief table does not contain any belief conditions (i.e. it coincides with its basic belief table). Then if χ is any code call condition of agent c, the extended code call atom

$$\mathbf{in}(\langle \mathsf{c}, \chi, \mathbf{true} \rangle, \mathsf{a} : \mathit{belief\_table}())$$

corresponds to the belief atom

$$\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{c}, \chi).$$

However, this does not mean that we can just replace the latter expression by the former. The problem is that beliefs need not neccessarily be stored in the belief table. They can also be triggered by entries from the belief table and those from the belief semantics table. In fact, this was why we explicitly formulated condition 7. on compatibility with the belief semantics table. Also if the third entry in the belief table is present, the belief condition, then the first two entries of this triple specify a belief that must hold. Therefore we will use the additional function

$$\mathsf{a} : \mathit{bel\_ccc\_act}(\sigma),$$

which was introduced in Definition 5.6 on page 28 and thus *implement* belief atoms with extended code calls:

$$\mathbf{in}("\chi", \mathsf{a} : \mathit{bel\_ccc\_act}(\mathsf{c}))$$

What happens if the formula χ is not a code call, but again a belief formula, say $\boldsymbol{\mathcal{B}}_{\mathsf{c}}(\mathsf{d}, \chi')$? An expression of the form $\mathbf{in}(\boldsymbol{\mathcal{B}}_{\mathsf{c}}(\mathsf{d}, \chi'), \mathsf{a} : \mathit{bel\_ccc\_act}(\mathsf{c}))$ is not a wellformed formula in our framework (recall that $\mathsf{a} : \mathit{bel\_ccc\_act}(\sigma)$ returns a set of code call conditions and action status atoms but no belief formulae). In fact, even if it were, it would not help in reducing the belief atoms to something involving only extended code calls. Here is where the inductive definition of our transformation (we call it $\mathfrak{Trans}$ from now on) comes in. We map

$$\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{c}, \boldsymbol{\mathcal{B}}_{\mathfrak{c}}(\mathfrak{d}, \chi'))$$

to

$$\mathbf{in}("\chi'", \mathfrak{a} : bel\_ccc\_act([\mathfrak{c}, \mathfrak{d}])).$$

Our main theorem in this section states that there is indeed a uniform transformation $\mathfrak{Trans}$ from arbitrary meta agent programs (which can also contain nested beliefs) to agent programs such that the semantics are preserved:

$$\mathsf{Sem}(\mathcal{BP}) = \mathsf{Sem}(\mathfrak{Trans}(\mathcal{BP})) \tag{6}$$

for Sem being a semantics based on *feasible*, *rational* or *reasonable* belief status sets.

6.1 Definition ($\mathfrak{Trans}$)
For an agent $\mathfrak{a}$, we defined in Definition 4.3 on page 16 the maximal belief language $\mathcal{BL}^{\mathfrak{a}}_{\infty}$. We define the mapping

$$\mathfrak{Trans} : \mathcal{BL}^{\mathfrak{a}}_{\infty} \rightarrow \textit{Code Call Conditions of } \mathcal{S}^{\text{ext}}$$

by induction on the structure of the belief literal:

Level 0: *If bel_lit is a code call condition or an action status atom, then $\mathfrak{Trans}(bel\_lit) =_{def} bel\_lit$.*

Level 1: *If bel_lit has the form $(\neg)\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \phi)$ where ϕ is a code call condition or an action status atom, then*

1. $\mathfrak{Trans}(\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \phi)) \mapsto \mathbf{in}("\phi", \mathfrak{a} : bel\_ccc\_act(\mathfrak{b}))$,
2. $\mathfrak{Trans}(\neg\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \phi)) \mapsto \mathbf{not\_in}("\phi", \mathfrak{a} : bel\_ccc\_act(\mathfrak{b}))$,

Level $n+1$: *If bel_lit has the form $(\neg)\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \phi)$ where ϕ is of level n, then we define $\mathfrak{Trans}(\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \phi))$ by*

$$\begin{cases} \mathbf{in}("\chi", \mathfrak{a} : bel\_ccc\_act([\mathfrak{b}, \rho])) & \textit{if } \mathfrak{Trans}(\phi) = \mathbf{in}("\chi", \mathfrak{a} : bel\_ccc\_act([\rho])) \\ \mathbf{in}("\chi", \mathfrak{a} : bel\_ccc\_act([\mathfrak{b}, -\rho])) & \textit{if } \mathfrak{Trans}(\phi) = \mathbf{not\_in}("\chi", \mathfrak{a} : bel\_ccc\_act([\rho])) \end{cases}$$

and we define $\mathfrak{Trans}(\neg\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \phi))$ by

$$\begin{cases} \mathbf{not\_in}("\chi", \mathfrak{a} : bel\_ccc\_act([\mathfrak{b}, \rho])) & \textit{if } \mathfrak{Trans}(\phi) = \mathbf{in}("\chi", \mathfrak{a} : bel\_ccc\_act(\rho)) \\ \mathbf{not\_in}("\chi", \mathfrak{a} : bel\_ccc\_act([\mathfrak{b}, -\rho])) & \textit{if } \mathfrak{Trans}(\phi) = \mathbf{not\_in}("\chi", \mathfrak{a} : bel\_ccc\_act(\rho)) \end{cases}$$

Linear Extension to $\mathcal{BL}^{\mathfrak{a}}_{\infty}$: *Up to now $\mathfrak{Trans}$ is only defined on belief literals, not for arbitrary belief formulae (which can be arbitrary conjunctions of belief literals (see Definition 4.3 on page 16). However we can easily extend $\mathfrak{Trans}$ so that it respects & by viewing the belief literals as a basis a naturally induced vector space. This new $\mathfrak{Trans}$ is then the uniquely determined homomorphism which coincides with the $\mathfrak{Trans}$ just defined.*

For a belief status set $\mathcal{BS}$ we denote by $\mathfrak{Trans}^{action}(\mathcal{BS})$ the subset of all action status atoms in $\mathcal{BS}$. This is exactly the status set as defined in Definition A.1 for agent programs without beliefs.

For a belief status set $\mathcal{BS}$ and an agent $\mathtt{b} \in \mathsf{A}$, we also define:

$$
\begin{array}{lll}
\mathfrak{Trans}^{state}(\mathcal{BS}, \mathtt{b}) & =_{def} & \{\chi \mid \boldsymbol{B}_{\mathfrak{a}}(\mathtt{b}, \chi) \in \mathcal{BS} \textit{ and } \chi \textit{ is a code call condition}\} \\
\mathfrak{Trans}^{action}(\mathcal{BS}, \mathtt{b}) & =_{def} & \{\mathbf{M}\alpha(\vec{t}) \mid \boldsymbol{B}_{\mathfrak{a}}(\mathtt{b}, \mathbf{M}\alpha(\vec{t})) \in \mathcal{BS}, \textit{ where } \mathbf{M} \in \{\mathbf{O}, \mathbf{W}, \mathbf{P}, \mathbf{F}, \mathbf{Do}\}\}.
\end{array}
$$

As in Definition 5.3 on page 27, these definitions are easily extended to arbitrary sequences σ instead of just $\mathtt{b}$.

This transformation $\mathfrak{Trans}$ maps all belief literals into extended code call conditions and will be used in the following to map any set containing belief literals (like belief status sets or meta agent programs) into one without belief literals but containing extended code calls. Also $\mathfrak{Trans}(\mathcal{BP})$ naturally defines an agent program without beliefs and thus we can use an existing implementation for agent programs to compute them.

Although the mapping $\mathfrak{Trans}$ is very simple, some more work is needed in order to get the above claimed equivalence result. Namely, in the definition of a feasible belief status set we have explicitly required the compatibility with the belief table and the belief semantics table (see Definitions 5.14 on page 34 and 5.16 on page 35). If we use $\mathfrak{Trans}$ to get rid of all belief atoms in a belief status set by transforming them into code calls, then we need to formulate similar conditions in terms of code calls. Otherwise we cannot expect a strong equivalence result to hold. The following picture may help to clarify the problem:

$$
\begin{tikzcd}
\mathcal{BP} \arrow[r, "\mathfrak{Trans}"] & \mathcal{P} \\
\mathcal{BS} \arrow[u, "\mathsf{Sem}^{\mathrm{new}}"', "\substack{\text{Compatible with}\\ \text{- Belief Semantics}\\ \text{- Belief Table}}"] \arrow[r, "\mathfrak{Trans}"] & S \arrow[u, "???"', "\mathsf{Sem}^{\mathrm{old}}"]
\end{tikzcd}
$$

It will be easy to show that if the conditions on the left side are fulfilled, and $\mathcal{BS}$ belongs to the semantics Sem of $\mathcal{BP}$, then S belongs to the semantics Sem of $\mathcal{P}$. But in order to reduce the semantics of meta agent programs to those of agent programs we must also have the converse, namely that all S's of $\mathcal{P}$ on the right hand side are induced by $\mathcal{BS}$'s on the left hand side. Such a result can only hold if we have corresponding conditions (indicated by "???" in the above diagram) on the right hand side.

The way we solve this problem is

1. to extend the original set of integrity constraints $\mathcal{IC}$ by a new constraint which expresses the compatibility with the belief semantics table using the new functions now available in $\mathcal{S}^{\mathrm{ext}}$,

2. to add a new condition (which cannot be expressed as an integrity constraint) which ensures the compatibility with the belief table.

As to 1. we denote by $\mathcal{IC}^{\text{ext}}$ the set $\mathcal{IC}$ of original integrity constraints augmented with the following extended integrity constraints (one for each agent $\mathfrak{b} \in \mathsf{A}$):

$$\begin{array}{ll}
\mathbf{in}(\langle \mathfrak{b}, \mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{b}} \rangle, \mathfrak{a} : belief_sem_table()) & \& \\
\mathbf{in}(\text{"}\chi\text{"}, \mathfrak{a} : bel_ccc_act(\mathfrak{b})) & \& \\
\mathbf{in}(\mathbf{true}, \mathfrak{a} : bel_semantics(\mathfrak{b}, \chi, \chi')) & \\
\Rightarrow & \\
\mathbf{in}(\text{"}\chi'\text{"}, \mathfrak{a} : bel_ccc_act(\mathfrak{b})). &
\end{array}$$

Note that we assume for ease of notation that the formulae χ, χ' are just code call conditions. In general, they can be arbitrary belief formulae (as determined by $\mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{b}}$). In this case, we have to take their transformation as provided by $\mathfrak{Trans}$. To be more precise, we have to add the constraints:

$$\begin{array}{ll}
\mathbf{in}(\langle \mathfrak{b}, \mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{b}} \rangle, \mathfrak{a} : belief_sem_table()) & \& \\
\mathbf{in}(\mathbf{true}, \mathfrak{a} : bel_semantics(\mathfrak{b}, \chi, \chi')) & \\
\mathfrak{Trans}(\chi) & \& \\
\Rightarrow & \\
\mathfrak{Trans}(\chi') &
\end{array}$$

As to 2. we require the following condition

Closure: Let the state $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfy $\mathbf{in}(\langle \mathfrak{b}, \phi, \chi_{\mathcal{B}} \rangle, \mathfrak{a} : belief\_table())$ as well as $\mathfrak{a}\_\mathsf{part}(\chi_{\mathcal{B}})$ and let $\mathfrak{Trans}^{\text{state}}(\mathcal{B}S, \mathfrak{b})$ satisfy $\mathfrak{b}\_\mathsf{part}(\chi_{\mathcal{B}})$. Let further ϕ be a code call condition or an action status atom,

1. If ϕ is a code call condition or an action status atom, then $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies $\mathbf{in}(\phi, \mathfrak{a} : bel\_ccc\_act(\mathfrak{b}))$.
2. If ϕ is of the form $\boldsymbol{\mathcal{B}}_{\mathfrak{b}}(\mathsf{c}, \phi')$, where ϕ' is a code call condition or an action status atom, then $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies $\mathbf{in}(\text{"}\phi'\text{"}, \mathfrak{a} : bel\_ccc\_act([\mathfrak{b}, \mathsf{c}]))$.
3. More generally, if ϕ is a nested belief atom, then we can associate with this atom a sequence $[\rho]$ (as introduced in Definition 5.2 on page 26) and we require that $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies $\mathbf{in}(\text{"}\phi'\text{"}, \mathfrak{a} : bel\_ccc\_act([\rho]))$.

Thus we end up up with the following picture:

$$\begin{array}{rcccl}
 & \mathcal{B}\mathcal{P} & \xrightarrow{\mathfrak{Trans}} & \mathcal{P} & \\
\begin{array}{l}\text{Compatible with}\\ \text{- Belief Semantics}\\ \text{- Belief Table}\end{array} & \Big\uparrow \mathsf{Sem}^{\text{new}} & & \begin{array}{c}\mathcal{IC}^{\text{ext}}\\ \text{Closure}\end{array} \Big\uparrow \mathsf{Sem}^{\text{old}} & \\
 & \mathcal{B}S & \xrightarrow{\mathfrak{Trans}} & S &
\end{array}$$

The following theorem and its corollaries make the statement (6) precise.

6.2 Theorem (Implementing Belief Programs by Agent Programs)
Let $\mathcal{BP}$ be a meta agent program, $\mathfrak{a} \in \mathsf{A}$, $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ a state of agent $\mathfrak{a}$, $\mathcal{IC}$ a set of integrity constraints, and $\mathcal{AC}$ a set of action constraints for $\mathfrak{a}$.

If $\mathcal{B}S$ is a feasible belief status set of agent $\mathfrak{a}$ wrt. $\mathcal{BP}$, $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$, $\mathcal{IC}$ and $\mathcal{AC}$, then

1. $\mathfrak{Trans}^{action}(\mathcal{BS})$ *is a feasible status set of* $\mathfrak{Trans}(\mathcal{BP})$ *wrt.* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ *and* $\mathcal{IC}^{\text{ext}}$. *In addition* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ *satisfies* ***Closure***.

2. *for all sequences* σ: $\mathfrak{Trans}^{action}(\mathcal{BS},\sigma)$ *is a feasible status set wrt.* $\mathfrak{Trans}^{state}(\mathcal{BS},\sigma)$ *and* $\mathcal{P}^{\mathfrak{a}}(\sigma)$, *where* $\mathbf{in}(\mathcal{P}^{\mathfrak{a}}(\sigma), \mathfrak{a}: \mathit{action\_program}(\sigma))$ *is true in* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$.

Moreover, every feasible status set of $\mathfrak{Trans}(\mathcal{BP})$ *for a state* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ *and* $\mathcal{IC}^{\text{ext}}$ *where* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ *satisfies* ***Closure*** *is obtained in that way.*

Proof: We first show 1. and 2. Let $\mathcal{BS}$ be a feasible belief status set of agent $\mathfrak{a}$ wrt. $\mathcal{BP}$, $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$, $\mathcal{IC}$ and $\mathcal{AC}$. $\mathfrak{Trans}^{\text{action}}(\mathcal{BS})$ is certainly a status set of $\mathfrak{Trans}(\mathcal{BP})$: it consists just of certain action status atoms for $\mathfrak{Trans}(\mathcal{BP})$. To check feasibility of this set, we have to check *(1) closure under program rules*, *(2) deontic and action consistecy*, *(3) deontic and action closure* and *(4) state consistency.* But all these properties are immediate from the corresponding conditions for $\mathcal{BS}$ (see Definition 5.12 for *(1)*, Definition 5.7 for *(2)*, Lemmas 5.8 and 5.9 for *(3)*, state consistency is analogously to *(4)* defined, and note that $\mathfrak{Trans}^{\text{action}}(\mathcal{BS},\sigma)$ and $\mathfrak{Trans}^{\text{state}}(\mathcal{BS},\sigma)$ correspond to $\Pi^{\text{action}}_{\sigma}(\mathcal{BS})$ and $\Pi^{\text{state}}_{\sigma}(\mathcal{BS})$).

Why is $\mathcal{IC}^{\text{ext}}$ true and why does $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfy **Closure**? $\mathcal{IC}^{\text{ext}}$ follows by the *belief semantics compatibility* condition and **Closure** by the *belief table compatibility.*

Condition 2. is implied by *local feasibility.*

Now we have to prove the converse, namely that every feasible status set of $\mathfrak{Trans}(\mathcal{BP})$ for a state $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ and $\mathcal{IC}^{\text{ext}}$ where $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies **Closure** is obtained in that way. Let S be such a feasible status set. Then we reconstruct $\mathcal{BS}^{\text{new}}$ using the code calls $\mathfrak{a}: \mathit{bel\_ccc\_act}([\rho])$. Whenever $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies a code call atom

$$\mathbf{in}(\text{"}\chi\text{"}, \mathfrak{a}: \mathit{bel\_ccc\_act}([\mathfrak{b}, \rho]))$$

where "χ" is a code call atom of the form "$\mathbf{in}(\phi, :())$" or an action status atom, then we add $\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \chi)$ to $\mathcal{BS}^{\text{new}}$. Note that because of the **Closure** condition, such code call atoms must hold and satisfy (if retransformed to belief formulae) the belief table compatibilty condition. By construction, $\mathcal{BS}^{\text{new}}$ is a status set and the feasibility is guaranteed by the feasiblity of S and the conditions we have just mentioned. ∎

6.3 Corollary ($\mathfrak{Trans}(\mathcal{BP})$ is invariant under Rational and Reasonable Semantics)
If $\mathcal{BS}$ *is a rational (resp. reasonable) belief status set of agent* $\mathfrak{a}$ *wrt.* $\mathcal{BP}$, $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$, $\mathcal{IC}$ *and* $\mathcal{AC}$, *then*

1. $\mathfrak{Trans}^{action}(\mathcal{BS})$ *is a* rational *(resp.* reasonable*) status set of* $\mathfrak{Trans}(\mathcal{BP})$ *wrt.* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ *satisfying* ***Closure*** *and wrt.* $\mathcal{IC}^{\text{ext}}$,

2. *for all sequences* σ: $\mathfrak{Trans}^{action}(\mathcal{BS},\sigma)$ *is a rational (resp. reasonable) status set wrt.* $\mathfrak{Trans}^{state}(\mathcal{BS},\sigma)$ *and* $\mathcal{P}^{\mathfrak{a}}(\sigma)$, *where* $\mathbf{in}(\mathcal{P}^{\mathfrak{a}}(\sigma), \mathfrak{a}: \mathit{action\_program}(\sigma))$ *is true in* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$.

Moreover, every rational *(resp.* reasonable*) status set of* $\mathfrak{Trans}(\mathcal{BP})$ *for a state* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ *and* $\mathcal{IC}^{\text{ext}}$ *where* $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ *satisfies* ***Closure*** *is obtained in that way.*

Proof: We distinguish between rational and reasonable status sets. As the latter are based on the former we first consider rational sets.

Rational: Using Theorem 6.2, it only remains to prove that every $\mathcal{BS}$ which is locally coherent, compatible with $\mathbf{BT}^a$ and with $\mathbf{BSemT}^a$ satisfies:

$\mathcal{BS}$ is grounded wrt. $\mathcal{BP}$ *if and only if* $\mathfrak{Trans}(\mathcal{BS})$ is grounded wrt. $\mathfrak{Trans}(\mathcal{BP})$.

This equivalence is easily shown by comparing the operators given in Definition 5.10 for programs with beliefs and Definition A.4 for programs without. Note that the transformation $\mathfrak{Trans}$ ensures that all belief literals of $\mathcal{BP}$ are transformed into extended code call conditions and these code call conditions are taken care of by our conditions (**Closure** and $\mathcal{IC}^{\text{ext}}$). A detailed inspection shows that every application of $\mathbf{App}_{\mathcal{BP},\mathcal{O}_\mathcal{S}}(\mathcal{BS})$ corresponds exactly to an application of $\mathbf{App}_{\mathfrak{Trans}(\mathcal{BP}),\mathcal{O}_{\mathcal{S}^{\text{ext}}}}(\mathfrak{Trans}(\mathcal{BS}))$ and thus the result follows.

Reasonable: Here we have to show that applying $\mathfrak{Trans}()$ is compatible with the reduction operation:

$$\mathfrak{Trans}(\mathsf{red}^{\mathcal{BS}}(\mathcal{BP},\mathcal{O}_{\mathcal{S}^{\text{ext}}})) = \mathsf{red}^{\mathfrak{Trans}(\mathcal{BS})}(\mathfrak{Trans}(\mathcal{BP}),\mathcal{O}_{\mathcal{S}^{\text{ext}}}).$$

The result then follows immediately by the definition of reasonable status sets, which are based on rational sets for positive programs. The problem is therefore reduced to the former case.

That the condition above holds follows immediately from the very definition of red. As we have a one-one correspondence between the body atoms of $\mathcal{BP}$ and those of $\mathfrak{Trans}(\mathcal{BP})$, a rule in $\mathcal{BP}$ is removed *if and only if* the corresponding rule in $\mathfrak{Trans}(\mathcal{BP})$ is removed. ∎

7 Related Work

In this paper, we have provided a framework within which an agent may reason about the beliefs it has about other agents' states, beliefs and possible actions. Our framework builds upon classical logic programming results. As there has been considerable work on these areas, we try to relate our work with the most relevant of these works. We do not explicitly relate ordinary agent programs (Eiter, Subrahmanian, and Pick 1998) with other agent systems, as that has been done in great detail in (Eiter, Subrahmanian, and Pick 1998). Rather, we focus primarily on meta-reasoning capabilities of agents and compare maps with meta reasoning capabilities of other agent frameworks.

Kowalski and Sadri (1998) have developed an agent architecture that uses logical rules expressed in Horn clause-like syntax, to encode agent behavior—both rational and reactive. The reactive agent rules are of the form

$$\alpha \quad \leftarrow \quad \text{condition}$$

where α is an action, and the condition in the body of the rule is a logical condition. Rationality is captured through integrity constraints. In the current language of (Kowalski and Sadri 1998), there seems to be no obvious support for meta-reasoning, though no doubt it could be encoded in, via some use of the metalogical *demo* predicate (Kowalski 1995).

M. Schroeder () have shown how extended logic programming may be used to specify the behavior of a diagnostic agent. They propose an architecture that supports cooperation between multiple diagnostic agents. Issues of interest arise when conflicting diagnoses are hypothesized by different agents. Their architecture consists of a knowledge base implemented by an extended logic program (Alferes and Pereira 1994), and inference machine that embodies the REVISE algorithm (C.V. Damasio and Pereira 1994) for eliminating contraditions, and a control layer. No meta-reasoning issues are brought up explicitly in this work.

Concurrently with our effort, M. Martelli and Zini (1998, M. Martelli and Zini (1997) have developed a logic programming based framework called CaseLP that may be used to implement multiagent applications by building on top of existing software. As in our work, agents have states, and states are changed by the agents' actions, and the behavior of an agent is encoded through rules. No meta-reasoning issues are brought up explicitly in this work.

Morgenstern (1990) was one of the first to propose a formal extension of auto-epistemic logic to deal with multiagent reasoning. She extended auto-epistemic logic (Moore 1985) with belief modalities indexed by agent names. She proposed a concept of expansions for such theories.

The *Procedural Reasoning System* (*PRS*) is one of the best known multiagent construction system that implements BDI agents (BDI stands for "Belief, Desires, Intentionality") (d'Inverno, Kinny, Luck, and Wooldridge 1997). This framework has led to several interesting applications including a practical, deployed application called *OASIS* for air traffic control in Sydney, Australia. The theory of *PRS* is captured through a logic based development, in (Rao and Georgeff 1991).

Gmytrasiewicz and Durfee (1992) have developed a logic of knowledge and belief to model multiagent coordination. Their framework permits an agent to reason not only about the world and its own actions, but also to simulate and model the behavior of other agents in the environment. In a separate paper (P. Gmytrasiewicz and Wehe. 1991), they show how one agent can reason with a probabilistic view of the behavior of other agents so as to achieve coordination. This is good work.

There are some significant differences between our work and theirs. First, we focus on agents that are built on top of arbitrary data structures. Second, our agent meta-reasoning language is very general—an agent can decide, for instance, that it will reason only with level 1 nested beliefs—and hence, our framework allows different agents to pick the level of belief reasoning appropriate for them. Third, our action framework is very general as well, and meta-reasoning with *permitted*, *obligatory* and *forbidden* actions is novel. Fourth, our framework allows an agent to "plug in" different estimates of the semantics used by other agents.

Researchers in the distributed knowledge community have also conducted extensive research into how one agent reasons about its beliefs about other agents (and their beliefs). Fagin and Vardi (1986) present a multiagent modal logic where knowledge modalities are indexed by agent names. They provide a semantics for message passing in such an environment. However, their work is quite different from ours.

Conclusions

We have seen that the operating principles governing how an agent acts, may, in many applications, be based upon the agent's beliefs about other agents' states, beliefs, and possible courses of actions. In order to effectively support such applications, we have proposed the notions of belief tables, and belief semantics tables, culminating in the definition of a *Meta Agent Program*, or map. We have shown that our map framework is rich enough to encode fairly complex meta-reasoning needs, such as those arising in the context of the RAMP example.

We have developed a formal semantics for maps—in particular, if a particular map $\mathcal{BP}$ is associated with an agent $\mathfrak{a}$, and the current state of the agent is $\mathcal{O}_{\mathcal{S}}$, then we have indicated what constitutes a *feasible belief status set.* Such a set indicates not only what the agent's *permitted*, *obligatory* and *forbidden* actions are, but also specifies what the agent believes to be the *permitted*, *obligatory* and *forbidden* actions of *other agents* are. We have then refined the concept of a feasible belief status set to two more fine grained semantics—namely the rational belief status set semantics, and the reasonable belief status set semantics.

Finally, we have provided a transformation that takes as input, a map and converts it into an ordinary agent program, together with a slightly modified version of the integrity constraints and object state. The *feasible* (resp. *rational*, *reasonable*) belief status sets of the map are shown to be in a one-one correspondence with the belief-free *feasible* (resp. *rational*, *reasonable*) status sets of the transformed agent progam with modified integrity constraints and object state. This result is nontrivial, and makes it possible to implement maps through a computation engine for *feasible* (resp. *rational*, *reasonable*) status sets of agent programs. We have currently developed a preliminary implementation of a computation engine for agent programs, and are currently refining it. Some sample screendumps of this engine may be seen at the following .

As beliefs that an agent $\mathfrak{a}$ may hold about another agent $\mathfrak{b}$ may be uncertain, we are currently extending the current work on maps to handle probabilistic modes of uncertainty. We are also extending maps so that agent $\mathfrak{a}$ can estimate what agent $\mathfrak{b}$ may do in the future, and to reason about how agent $\mathfrak{b}$'s beliefs may evolve in the future.

Acknowledgments

This work was supported by the Army Research Office under Grants DAAH-04-95-10174, DAAH-04-96-10297, DAAG-55-97-10047 and DAAH04-96-1-0398, by the Army Research Laboratory under contract number DAAL01-97-K0135 and by an NSF Young Investigator award IRI-93-57756.

References

Alferes, J. and L. Pereira (1994). Reasoning with Logic Programming. In *Springer Lecture Notes in Artificial Intelligence vol. 1111.*

Åquist, L. (1984). Deontic Logic. In D. Gabbay and F. Guenthner (Eds.), *Handbook of Philosophical Logic, Vol.II*, Chapter II.11, pp. 605–714. D. Reidel Publishing Company.

Arens, Y., C. Chee, C.-N. Hsu, and C. Knoblock (1993). Retrieving and Integrating Data From Multiple Information Sources. *International Journal of Intelligent Cooperative Information Systems 2*(2), 127–158.

Arisha, K., S. Kraus, F. Ozcan, R. Ross, and V.S.Subrahmanian (1997, November). IMPACT: The Interactive Maryland Platform for Agents Collaborating Together. Submitted for publication.

Bayardo *et. al.*, R. (1997). Infosleuth: Agent-based Semantic Integration of Information in Open and Dynamic Environments. In *ACM SIGMOD Conf. on Management of Data*, Arizona, USA.

Benton, J. and V. Subrahmanian (1994). Using Hybrid Knowledge Bases for Missile Siting Problems. In I. C. Society (Ed.), *Proceedings of the Conference on Artificial Intelligence Applications*, pp. 141–148.

Bonatti, P., S. Kraus, J. Salinas, and V. Subrahmanian (1998). Data Security in Heterogenous Agent Systems. In M. Klusch (Ed.), *Cooperative Information Agents*, pp. 290–305. Springer.

Brink, A., S. Marcus, and V. Subrahmanian (1995). Heterogeneous Multimedia Reasoning. *IEEE Computer 28*(9), 33–39.

Candan, K., B. Prabhakaran, and V. Subrahmanian (1996, November). CHIMP: A Framework for Supporting Multimedia Document Authoring and Presentation. In *ACM Multimedia Conference*, Boston, MA.

Cattell *et.al.*, R. (1997). *The Object Database Standard: ODMG-93.* Morgan Kaufmann.

Chawathe, S., H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom (1994, October). The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *IPSJ Conference*, Tokyo, Japan. Also available via anonymous FTP from host db.stanford.edu, file /pub/chawathe/1994/tsimmis-overview.ps.

C.V. Damasio, W. N. and L. Pereira (1994). An Extended Logic Programming System for Revising Knowledge Bases. In *Proceedings of KR-94*. Morgan Kaufman.

d'Inverno, M., D. Kinny, M. Luck, and M. Wooldridge (1997). A Formal Specification of dMARS. In *Intl. Workshop on Agent Theories, Architectures, and Languages*, Providence, RI, pp. 146–166.

Eiter, T., V. Subrahmanian, and G. Pick (1998). Heterogenous Active Agents. Technical report, University of Maryland, Dept. of CS.

Fagin, R. and M. Vardi (1986). Knowledge and Implicit Knowledge in a Distributed Environment. In *Proc. 1986 Conf on Theoretical Aspects of Reasoning about Knowledge*, pp. 187–206. Morgan Kaufman.

Foltz, P. W. and S. T. Dumais (1992). Personalized information delivery: An analysis of filtering methods. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems – Posters and Short Talks*, Posters: Designing for Use.

Gmytrasiewicz, P. and E. Durfee (1992). A logic of knowledge and belief for recursive modeling. In *Proceedings of AAAI-92*, pp. 628–634. Morgan Kaufman.

Goldberg, D., D. Nichols, B. Oki, and D. Terry (1992, September). Using collaborative filtering to weave an information tapestry. Technical Report CSL-92-10, parc, Palo Alto, California.

Hindriks, K., F. de Boer, W. van der Hoek, and J. Meyer (1997). Formal Semantics of an Abstract Agent Programming Language. In *Intl. Workshop on Agent Theories, Architectures, and Languages*, Providence, RI, pp. 204–218.

Ishizaki, S. (1997). Multiagent Model of Dynamic Design: Visualization as an Emergent Behavior of Active Design Agents. In M. Huhns and M. Singh (Eds.), *Readings in Agents*, pp. 172–179. Morgan Kaufmann Press.

Kanger, S. (1972). Law and Logic. *Theoria 38.*

Kowalski, R. (1995). *Using metalogic to reconcile reactive with rational agents.* MIT Press.

Kowalski, R. and F. Sadri (1998). Towards a unified agent architecture that combines rationality with reactivity. draft manuscript.

Labrou, Y. and T. Finin (1997). Semantics for an Agent Communication Language. In *Intl. Workshop on Agent Theories, Architectures, and Languages*, Providence, RI, pp. 199–203.

Lu, J., A. Nerode, and V. Subrahmanian (1996). Hybrid Knowledge Bases. *IEEE Transactions on Knowledge and Data Engineering 8*(5), 773–785.

M. Martelli, V. M. and F. Zini (1997). Caselp: a complex application specification environment based on logic programming. In *Proc. of ICLP'97 Post Conference Workshop on Logic Programming and Multi-Agents*, Leuven, Belgium, pp. 35–50.

M. Martelli, V. M. and F. Zini (1998). Towards multi-agent software prototyping. In *Proc. of The Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM98)*, London, UK, pp. 331–354.

M. Schroeder and I. de Almeida Mora and L. M. Pereira (1997). A deliberative and reactive diagnosis agent based on logic programming. In J. Muller, M. Wooldridge, and N. Jennings (Eds.), *Intelligent Agents III: Lecture Notes in Artificial Intelligence Vol. 1193*, pp. 293–307. Springer Verlag.

Meyer, J.-J. C. and R. Wieringa (Eds.) (1993). *Deontic Logic in Computer Science.* Chichester et al: Wiley & Sons.

Moore, R. (1985). Semantical Considerations on Nonmonotonic Logics. *Artificial Intelligence 25*, 75–94.

Morgenstern, L. (1990). A formal theory of multiple agent nonmonotonic reasoning. In *Proceedings of AAAI-90*, pp. 538–544.

P. Gmytrasiewicz, E. D. and D. Wehe. (1991). A Decision-Theoretic Approach to Coordinating Multiagent Interactions. In *Proceedings of IJCAI 1991*, pp. 62–68. Morgan Kaufman.

Rao, A. and M. Georgeff (1991). Modeling Rational Agents within a BDI-Architecture. In J. F. Allen, R. Fikes, and E. Sandewall (Eds.), *Proceedings of the International Conference on Knowledge Representation and Reasoning*, Cambridge, MA, pp. 473–484. Morgan Kaufmann.

Rosenschein, S. (1985). Formal Theories of Knowledge in AI and Robotics. *New Generation Computing 3*(4), 345–357.

Rosenschein, S. and L. Kaelbling (1995). A Situated View of Representation and Control. *Artificial Intelligence 73*, 149–173.

Rus, D., R. Gray, and D. Kotz (1997). Transportable Information Agents. In M. Huhns and M. Singh (Eds.), *Readings in Agents*, pp. 283–291. Morgan Kaufmann Press.

Sheth, B. and P. Maes (1993, Mrach). Evolving agents for personalized information filtering. In *Proceedings of the 9th Conference on Artificial Intelligence for Applications (CAIA'93)*, pp. 345–352. IEEE Computer Society Press.

Shoham, Y. (1993). Agent Oriented Programming. *Artificial Intelligence 60*, 51–92.

Siegal, J. (1996). *CORBA Fundementals and Programming*. New York: John Wiley and Sons.

Sta, J.-D. (1993). Information filtering: A tool for communication between researchers. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems – Adjunct Proceedings*, Short Papers (Posters): Help and Information Retrieval, pp. 177–178.

Ullman, J. D. (1989). *Principles of Database and Knowledge Base Systems*. Computer Science Press.

Wilder, F. (1993). *A Guide to the TCP/IP Protocol Suite*. Artech House.

A Agent Programs without Beliefs

A.1 Feasible, Rational and Reasonable Semantics

A.1 Definition (Status Set)
A status set *is any set S of ground action status atoms over $\mathcal{S}$. For any operator $\mathit{Op} \in \{\mathbf{P}, \mathbf{Do}, \mathbf{F}, \mathbf{O}, \mathbf{W}\}$, we denote by $\mathit{Op}(S)$ the set $\mathit{Op}(S) = \{\alpha \mid \mathit{Op}(\alpha) \in S\}$.*

A.2 Definition (Deontic and Action Consistency)
A status set S is called deontically consistent, *if, by definition, it satisfies the following rules for any ground action α:*

- *If $\mathbf{O}\alpha \in S$, then $\mathbf{W}\alpha \notin S$*
- *If $\mathbf{P}\alpha \in S$, then $\mathbf{F}\alpha \notin S$*
- *If $\mathbf{P}\alpha \in S$, then $\mathcal{O}_{\mathcal{S}} \models \exists^* \mathit{Pre}(\alpha)$, where $\exists^* \mathit{Pre}(\alpha)$ denotes the existential closure of $\mathit{Pre}(\alpha)$, i.e., all free variables in $\mathit{Pre}(\alpha)$ are governed by an existential quantifier. This condition means that the action α is in fact executable in the state $\mathcal{O}_{\mathcal{S}}$.*

A status set S is called action consistent, *if $S, \mathcal{O}_{\mathcal{S}} \models \mathcal{AC}$ holds.*

Besides consistency, we also wish that the presence of certain atoms in S entails the presence of other atoms in S. For example, if $\mathbf{O}\alpha$ is in S, then we expect that $\mathbf{P}\alpha$ is also in S, and if $\mathbf{O}\alpha$ is in S, then we would like to have $\mathbf{Do}\alpha$ in S. This is captured by the concept of deontic and action closure.

A.3 Definition (Deontic and Action Closure)
The deontic closure *of a status S, denoted $\mathbf{D\text{-}Cl}(S)$, is the closure of S under the rule*

If $\mathbf{O}\alpha \in S$, *then* $\mathbf{P}\alpha \in S$

where α is any ground action. We say that S is deontically closed, *if $S = \mathbf{D\text{-}Cl}(S)$ holds.*

The action closure *of a status set S, denoted $\mathbf{A\text{-}Cl}(S)$, is the closure of S under the rules*

If $\mathbf{O}\alpha \in S$, *then* $\mathbf{Do}\alpha \in S$

If $\mathbf{Do}\alpha \in S$, *then* $\mathbf{P}\alpha \in S$

where α is any ground action. We say that a status S is action-closed, if $S = \mathbf{A\text{-}Cl}(S)$ holds.

The following straightforward results shows that status sets that are action-closed are also deontically closed, i.e.

A.4 Definition (Operator $\mathbf{App}_{\mathcal{P},\mathcal{O}_\mathcal{S}}(S)$)
Suppose $\mathcal{P}$ is an agent program, and $\mathcal{O}_\mathcal{S}$ is an agent state. Then, $\mathbf{App}_{\mathcal{P},\mathcal{O}_\mathcal{S}}(S)$ is defined to be the set of all ground action status atoms A such that there exists a rule in P having a ground instance of the form $r : A \leftarrow L_1, \dots, L_n$ such that

1. $B^+_{as}(r) \subseteq S$ *and* $\neg.B^-_{as}(r) \cap S = \emptyset$, *and*
2. *every code call* $\chi \in B^+_{cc}(r)$ *succeeds in* $\mathcal{O}_\mathcal{S}$, *and*
3. *every code call* $\chi \in \neg.B^-_{cc}(r)$ *does not succeed in* $\mathcal{O}_\mathcal{S}$, *and*
4. *for every atom* $\mathit{Op}(\alpha) \in B^+(r) \cup \{A\}$ *such that* $\mathit{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$, *the action α is executable in state* $\mathcal{O}_\mathcal{S}$.

Note that part (4) of the above definition only applies to the "positive" modes $\mathbf{P}, \mathbf{O}, \mathbf{Do}$. It does not apply to atoms of the form $\mathbf{F}\alpha$ as such actions are not executed, nor does it apply to atoms of the form $\mathbf{W}\alpha$, because execution of an action might be (vacuously) waived, if its prerequisites are not fulfilled.

Our approach is to base the semantics of agent programs on consistent and closed status sets. However, we have to take into account the rules of the program as well as integrity constraints. This leads us to the notion of a feasible status set.

A.5 Definition (Feasible Status Set)
Let $\mathcal{P}$ be an agent program and let $\mathcal{O}_\mathcal{S}$ be an agent state. Then, a status set S is a feasible status set *for $\mathcal{P}$ on $\mathcal{O}_\mathcal{S}$, if the following conditions hold:*

(S1): *(closure under the program rules)* $\boldsymbol{App}_{\mathcal{P},\mathcal{O}_{\mathcal{S}}}(S) \subseteq S$;

(S2) *(deontic and action consistency)* S *is deontically and action consistent;*

(S3) *(deontic and action closure)* S *is action closed and deontically closed;*

(S4) *(state consistency)* $\mathcal{O}'_{\mathcal{S}} \models \mathcal{IC}$, *where* $\mathcal{O}'_{\mathcal{S}} = apply(\mathbf{Do}(S), \mathcal{O}_{\mathcal{S}})$ *is the state which results after taking all actions in* $\mathbf{Do}(S)$ *on the state* $\mathcal{O}_{\mathcal{S}}$.

A.6 Definition (Groundedness; Rational Status Set)
A status set S *is* grounded, *if there exists no status set* $S' \neq S$ *such that* $S' \subseteq S$ *and* S' *satisfies conditions* $(S1)$–$(S3)$ *of a feasible status set.*

A status set S *is a* rational status set, *if* S *is a feasible status set and* S *is grounded.*

A.7 Definition (Reasonable Status Set)
Let $\mathcal{P}$ *be an agent program, let* $\mathcal{O}_{\mathcal{S}}$ *be an agent state, and let* S *be a status set.*

1. *If* $\mathcal{P}$ *is a positive agent program, then* S *is a* reasonable status set *for* $\mathcal{P}$ *on* $\mathcal{O}_{\mathcal{S}}$, *if and only if* S *is a rational status set for* $\mathcal{P}$ *on* $\mathcal{O}_{\mathcal{S}}$.
2. *The reduct of* $\mathcal{P}$ *w.r.t.* S *and* $\mathcal{O}_{\mathcal{S}}$, *denoted by* $red^S(\mathcal{P}, \mathcal{O}_{\mathcal{S}})$, *is the program which is obtained from the ground instances of the rules in* $\mathcal{P}$ *over* $\mathcal{O}_{\mathcal{S}}$ *as follows.*
   (a) *First, remove every rule* r *such that* $B^-_{as}(r) \cap S \neq \emptyset$;
   (b) *Remove all atoms in* $B^-_{as}(r)$ *from the remaining rules.*

   Then S *is a* reasonable status set *for* $\mathcal{P}$ *w.r.t.* $\mathcal{O}_{\mathcal{S}}$, *if it is a reasonable status set of the program* $red^S(\mathcal{P}, \mathcal{O}_{\mathcal{S}})$ *with respect to* $\mathcal{O}_{\mathcal{S}}$.

B Agents in RAMP

B.1 Helicopter Agent

B.1.1 Code Calls

1. Change flying altitude to `Altitude` (`0` to `Maximum` altitude):

   `Heli`: *SetAltitude*(`Altitude`) $\rightarrow$ `Boolean`

2. Get current altitude:

   `Heli`: *GetAltitude*(`now`) $\rightarrow$ `Altitude`

3. Change flying speed to `Speed` (`0` to `Maximum` speed):

   `Heli`: *SetSpeed*(`Speed`) $\rightarrow$ `Boolean`

4. Get current speed:

Heli: $GetSpeed$(now) $\rightarrow$ Speed

5. Change flying heading to Heading (0 to 360):

   Heli: $SetHeading$(Heading) $\rightarrow$ Boolean

6. Get current heading:

   Heli: $GetHeading$(now) $\rightarrow$ Heading

7. Aim the gun at the 3D point given by Position:

   Heli: Aim(Position) $\rightarrow$ Boolean

8. Fire the gun using the current aim:

   Heli: $Fire$(now) $\rightarrow$ Boolean

9. Determine the current position in space:

   Heli: $GetPosition$(now) $\rightarrow$ 3DPoint

10. Compute heading to fly from 2D point Src to 2D point Dst:

    Heli: $ComputeHeading$(Src, Dst) $\rightarrow$ Heading)

11. Compute the distance between two 3D points:

    Heli: $ComputeDistance$(X, Y) $\rightarrow$ Distance

12. Retrieve the maximum range for the gun:

    Heli: $GetMaxGunRange$(now) $\rightarrow$ Distance

B.1.2 Actions

1. *Fly* from 3D point From to 3D point To at altitude Altitude and speed Speed

   Fly(From ,To ,Altitude ,Speed)
   $Pre(Fly)$: **in**(From, Heli: $GetPosition$(now))
   $Del(Fly)$: **in**(From, Heli: $GetPosition$(now))
   $Add(Fly)$: **in**(To, Heli: $GetPosition$(now + 1))

2. $FlyRoute(path)$ Path given as a sequence of triples consisting of: a 3D point, altitude, and speed

   $FlyRoute$(Path)
   $Pre(FlyRoute)$: **in**(Path(0).Position, Heli: $GetPosition$(now))
   $Del(FlyRoute)$: **in**(Path(0).Position, Heli: $GetPosition$(now))
   $Add(FlyRoute)$: **in**(Path(Path.Count).Position, Heli: $GetPosition$(now + 1))

3. Attack vehicle at position `Position` in space

   Attack(Position)
   Pre(*Attack*): **in**(MyPosition, Heli : *GetPosition*(now)) &
   in(Distance, Heli : *ComputeDistance*(MyPosition, Position, now)) &
   in(MaxRange, Heli : *GetMaxGunRange*(now)) &
   Distance < MaxRange
   Del(*Attack*): {}
   Add(*Attack*): {}

B.1.3 Integrity Constraints

in(S, Heli : *GetSpeed*(now)) & S $<$ MaxSpeed
in(A, Heli : *GetAltitude*(now)) & A $<$ MaxAltitude

B.1.4 Action Constraints

{ *Fly*(X1,Y1,A1,S1), *Fly*(X2,Y2,A2,S2)}
$\hookleftarrow$ X1 != X2 Or Y1 != Y2 Or A1 != A2 Or S1 != S2
{ *Attack*(P)} $\hookleftarrow$ **in**(P, Heli : *GetPosition*(now))

B.2 Tank Agent

B.2.1 Code Calls

1. Drive forward at speed `Speed` (`0` to `Max` speed)

   Tank : *GoForward*(Speed) $\rightarrow$ Boolean

2. Drive backward at speed `Speed` (`0` to `Max` speed)
   Tank : *GoBackward*(Speed)

3. Turn left by `Degrees` degrees (`0` to `360`)
   Tank : *TurnLeft*(Degrees)

4. Turn right by `Degrees` degrees (`0` to `360`)
   Tank : *TurnRight*(Degrees)

5. Determine current position in 2D

   Tank : *GetPosition*(now) $\rightarrow$ 2DPoint

6. Get current heading

   Tank : *GetHeading*(now) $\rightarrow$ Heading

7. Aim the gun at 3D point `Point`

   Tank : *Aim*(Point) $\rightarrow$ Boolean

8. Fire the gun using the current aim

   $\mathsf{Tank}: \mathit{Fire}(\texttt{now}) \rightarrow \texttt{Boolean}$

9. Compute the distance between two 2D points

   $\mathsf{Tank}: \mathit{ComputeDistance}(\texttt{X})\texttt{Y} \rightarrow \texttt{Distance}$

10. Retrieve the maximum range for the gun

    $\mathsf{Tank}: \mathit{GetMaxGunRange}(\texttt{now}) \rightarrow \texttt{Distance}$

B.2.2 Actions

1. Drive from to 2D point `From` to 2D point `To` at speed `Speed`

   $\mathit{Drive}(\texttt{From,To,Speed})$
   $\mathit{Pre}(\mathit{Drive}):\ \mathbf{in}(\texttt{From}, \mathsf{Tank}: \mathit{GetPosition}(\texttt{now}))$
   $\mathit{Del}(\mathit{Drive}):\ \mathbf{in}(\texttt{From}, \mathsf{Tank}: \mathit{GetPosition}(\texttt{now}))$
   $\mathit{Add}(\mathit{Drive}):\ \mathbf{in}(\texttt{To}, \mathsf{Tank}: \mathit{GetPosition}(\texttt{now}+1))$

2. Drive route `Route` given as a sequence of 2D points at speed `Speed`

   $\mathit{DriveRoute}(\texttt{Route,Speed})$
   $\mathit{Pre}(\mathit{DriveRoute}):\ \mathbf{in}(\texttt{Route(0).Position}, \mathsf{Tank}: \mathit{GetPosition}(\texttt{now}))$
   $\mathit{Del}(\mathit{DriveRoute}):\ \mathbf{in}(\texttt{Route(0).Position}, \mathsf{Tank}: \mathit{GetPosition}(\texttt{now}))$
   $\mathit{Add}(\mathit{DriveRoute}):\ \mathbf{in}(\texttt{Route(Route.Count).Position}, \mathsf{Tank}: \mathit{GetPosition}(\texttt{now}+1))$

3. Attack vehicle at position `Position` in space

   $\mathit{Attack}(\texttt{Position})$
   $\mathit{Pre}(\mathit{Attack}):\ \mathbf{in}(\texttt{MyPosition}, \mathsf{Tank}: \mathit{GetPosition}(\texttt{now}))\ \ \&$
   $\quad \mathbf{in}(\texttt{Distance}, \mathsf{Tank}: \mathit{ComputeDistance}(\texttt{MyPosition},\ \texttt{Position}, \texttt{now}))\ \&$
   $\quad \mathbf{in}(\texttt{MaxRange}, \mathsf{Tank}: \mathit{GetMaxGunRange}(\texttt{now}))\ \ \&$
   $\quad \texttt{Distance < MaxRange}$
   $\mathit{Del}(\mathit{Attack}):\ \{\}$
   $\mathit{Add}(\mathit{Attack}):\ \{\}$

B.3 Terrain Route Planning Agent

B.3.1 Code Calls

1. Sets current map to `Map`

   $\mathsf{Route}: \mathit{UseMap}(\texttt{Map}) \rightarrow \texttt{Bool}$

2. Compute a route plan on the current map for a vehicle of type `VehicleType` from `SourcePoint` to `DestinationPoint` given in 2D. Returns a route plan as a sequence of points in plane.

$\text{Route}: \mathit{GetPlan}(\texttt{SourcePoint}, \texttt{DestinationPoint}, \texttt{VehicleType})$
$\quad \rightarrow \texttt{SequenceOf2DPoints}$

3. Given `SourcePoint` and `DestinationPoint` on the current map, determine the likely routes of a vehicle of type `VehicleType` whose initial route segment is `Route`, given as a sequence of points in the plane It returns a sequence of route-probability pairs.

   $\text{Route}: \mathit{GroundPlan}(\texttt{SourcePoint}, \texttt{DestinationPoint}, \texttt{VehicleType}, \texttt{Route})$
   $\quad \rightarrow (\texttt{Route}, \texttt{Probability})$

4. Compute a flight plan on the current map from `SourcePoint` to `DestinationPoint` given in 3D. Returns a flight plan as a sequence of points in space

   $\text{Route}: \mathit{FlightPlan}(\texttt{SourcePoint}, \texttt{DestinationPoint})$
   $\quad \rightarrow \texttt{SequenceOf3DPoints}$

5. Determines whether two points are visible from each other on the given map. For example if a hill lies between the two points, they are not visible from each other. This is useful to determine whether an agent can see another agent or whether an agent can fire upon another agent.

   $\text{Route}: \mathit{Visible}(\texttt{Map}, \texttt{Point1}, \texttt{Point2}) \rightarrow \texttt{Boolean}$

B.3.2 Actions

1. Compute a route plan on map `Map` for a vehicle of type `VehicleType` from `SourcePoint` to `DestinationPoint` given in 2D.

   $\mathit{PlanRoute}(\texttt{Map,SourcePoint,DestinationPoint,VehicleType})$
   $\mathit{Pre}(\mathit{PlanRoute}): \texttt{SourcePoint != DestinationPoint}$
   $\mathit{Del}(\mathit{PlanRoute}): \{\}$
   $\mathit{Add}(\mathit{PlanRoute}): \textbf{in}(\textbf{true}, \text{Route}: \mathit{UseMap}(\texttt{Map}, \texttt{now}))\ \ \texttt{\&}$
   $\quad \textbf{in}(\texttt{Plan}, \text{Route}: \mathit{GetPlan}(\texttt{SourcePoint}, \texttt{DestinationPoint}, \texttt{VehicleType}, \texttt{now}))$

2. Given `SourcePoint` and `DestinationPoint` on map `Map` determine the likely routes of a vehicle of type `VehicleType` whose initial route segment is `Route`, given as a sequence of points in the plane

   $\mathit{EvaluateGroundPlan}(\texttt{Map,SourcePoint,DestinationPoint,VehicleType,Route})$
   $\mathit{Pre}(\mathit{EvaluateGroundPlan}): \texttt{SourcePoint != DestinationPoint}$
   $\mathit{Del}(\mathit{EvaluateGroundPlan}): \{\}$
   $\mathit{Add}(\mathit{EvaluateGroundPlan}): \textbf{in}(\textbf{true}, \text{Route}: \mathit{UseMap}(\texttt{Map}, \texttt{now}))\ \ \texttt{\&}$
   $\quad \textbf{in}(\texttt{RP}, \text{Route}: \mathit{GroundPlan}(\texttt{SourcePoint}, \texttt{DestinationPoint}, \texttt{VehicleType}, \texttt{Route}, \texttt{now}))$

3. Compute a flight plan on map `Map` from `SourcePoint` to `DestinationPoint` given in 3D.

   $\mathit{PlanFlight}(\texttt{Map,SourcePoint,DestinationPoint})$
   $\mathit{Pre}(\mathit{PlanFlight}): \texttt{SourcePoint != DestinationPoint}$
   $\mathit{Del}(\mathit{PlanFlight}): \{\}$
   $\mathit{Add}(\mathit{PlanFlight}): \textbf{in}(\textbf{true}, \text{Route}: \mathit{UseMap}(\texttt{Map}, \texttt{now}))\ \ \texttt{\&}$
   $\quad \textbf{in}(\texttt{Plan}, \text{Route}: \mathit{FlightPlan}(\texttt{SourcePoint}, \texttt{DestinationPoint}, \texttt{now}))$

B.4 Tracking Agent

This agent continuously scans the area for enemy vehicles. It maintains a list of enemy vehicles, assigning each an agent id. It tries to determine the vehicle type for each enemy vehicle. When it detects a new vehicle, it adds it to its list, together with its position. Since the tracking agent only keeps track of enemy vehicles which are on the ground, the position is in the plane. This could be for example an *AWACS* plane.

B.4.1 Code Calls

1. Get position for agent with id `AgentId` at time `Time` If time is in the past this is done by searching the database. If time is in the future this is done by `guessing` the position.

   $\mathsf{Track}: \mathit{GetPosition}(\texttt{AgentId}, \texttt{Time}) \rightarrow \texttt{2DPoint}$

2. Get the type of agent for agent with id `AgentId`. It returns the most likely vehicle type together with the probability

   $\mathsf{Track}: \mathit{GetTypeOfAgent}(\texttt{AgentId}) \rightarrow (\texttt{VehicleType}, \texttt{Probability})$

3. Return the list of all agents being tracked

   $\mathsf{Track}: \mathit{GetListOfAgents}(\texttt{now}) \rightarrow \texttt{ListOfAgentIdsF}$

B.5 Coordination Agent

B.5.1 Code Calls

1. Determine wether a vehicle of type `VehicleType1` at position `Position1` can attack a vehicle of type `VehicleType2` at position `Position2`. For example a tank is not able to attack a fighter plane unless it is on the ground.

   $$\mathsf{Coord}: \mathit{CanBeAttackedNow}((\texttt{VehicleType1}, \texttt{Position1}, \texttt{VehicleType2}, \texttt{Position2}))$$
   $$\rightarrow \texttt{Boolean}$$

2. Given an agent id for an enemy vehicle, determine the best position, time and route for an attack to be successful. Also return the estimated probability of success

   $$\mathsf{Coord}: \mathit{FindAttackTimeAndPosition}(\texttt{AgentId})$$
   $$\rightarrow (\texttt{Position}, \texttt{Time}, \texttt{Route}, \texttt{Probability})$$

3. Given a set of ids for friendly agents, compute a plan for a coordinated attack against the enemy agent with id `EnemyId`. The friendly agents participating in the coordinated attack are taken from the set `SetOfAgentIds`

   $$\mathsf{Coord}: \mathit{CoordinatedAttack}((\texttt{SetOfAgentIds}, \texttt{EnemyId}))$$
   $$\rightarrow \texttt{AttackPlan}$$

B.5.2 Actions

1. Given a set of ids for friendly agents, compute a plan for a coordinated attack against the enemy agent with id `EnemyId`. The friendly agents participating in the coordinated attack are taken from the set `SetOfAgentIds`.

   Attack(`SetOfAgentIds,EnemyId`)
   Pre(*Attack*): `SetOfAgentIds != {}`
   Del(*Attack*): $\{\}$
   Add(*Attack*): $\mathbf{in}(\mathtt{AP}, \mathrm{Coord} : \mathit{CoordinatedAttack}((\mathtt{SetOfAgentIds}, \mathtt{EnemyId}, \mathtt{now})))$