Meta-Agent Programs

Jürgen Dix [??], V.S. Subrahmanian [b] and George Pick [c]

[a] *The University of Manchester, Dept. of Computer Science, Oxford Road, Manchester, M13 9PL, UK. E-mail: dix@cs.man.ac.uk*

[b] *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*

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

Abstract

There are numerous applications where an agent a needs to reason about the beliefs of another agent, as well as about the actions that other agents may take. In (21) the concept of an agent program is introduced, and a language within which the operating principles of an agent can be declaratively encoded on top of imperative data structures is defined. In this paper 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 framework of (21; 19), so as to allow agents to perform 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 services (6; 4; 11; 36; 14), to mobile agents (45), to personalized visualization agents (12; 29), to agents that monitor newspapers, prioritize mail and the like (27; 24; 50; 47).

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 principles in (43) and (44). Declarative languages to encode such operating principles were proposed by Shoham (48) and (28). Recently,in (21; 19), the authors 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.

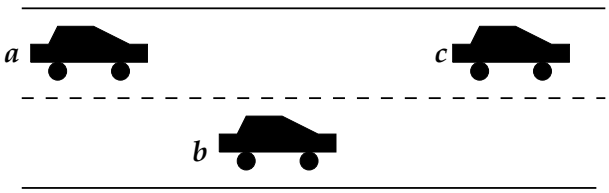

Figure 1. Autonomous Vehicle

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

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

Based on these two beliefs—the first which is about the state of vehicle $\mathtt{b}$, while the second is about the actions of vehicle $\mathtt{b}$—vehicle $\mathtt{a}$ 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 $\mathtt{a}$ may believe that vehicle $\mathtt{b}$ believes that vehicle $\mathtt{c}$ is about to shift to the right lane, etc.

In general, this very simple, everyday example shows that an agent may need to reason about other agents' state, beliefs, and potential actions. In most existing agent languages such as (48; 28; 21; 19), the notion of an agent state is general enough to include almost anything. However, the specific needs of agents such as vehicle $\mathtt{a}$ 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 (5; 21; 8; 19). In Section 3, we briefly overview the basic architecture of our *IMPACT* system, and quickly describe the decision making framework of (21; 19).
- The new contributions of this paper start in 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 a map as input, and converts it into an ordinary agent program, together with some integrity constraints, as defined in (21; 19). 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 the *feasible*, *rational*, and *reasonable* status sets of ordinary agent programs.

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* (7).

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 from various agents (e.g. the tracking agents) 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 (e.g. 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 has 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. This is doubly difficult, given the fact that the enemy might be explicitly trying to deceive the RAMP agents. 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, $0 \leq i \leq now$. 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}. The coordination agent may even allow for potential deception on the part of the enemy agent. For example, it might maintain probabilities of events of the form "asset A_i is the intended target of the enemy." As the enemy moves, these probabilities can be constantly updated by the agent.
- *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 22 on page 23). 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

Throughout this paper we use A to denote a finite set whose elements are called *agents*. Figure 2 on page 9 shows the different components of an agent. We briefly describe these components below—for a full description, the reader is referred to (21; 19).

Agent Software Code: Most existing work on agents has assumed a purely logical description of agents. However, a fundamental tenet of computer science is that when storing data, we must choose a data structure that "best" supports the expected accesses on that data by the user community in question. Thus, for agents to efficiently deliver services to users, they must be able to work with application specific data structures, rather than assuming that all data will be logically represented. Logic must "build" on top of such third part data structures to support different types of intelligent reasoning and decision making.

In addition, agent theorists must support the idea that third party packages be able to "play" in an agent environment. We expect many existing imperative programs to be "extended" to become agents (such as "agentization" procedure is described in (21; 19)), and as we want agent developers to have maximal flexibility in their choice of a data structure(s) for their application, we build our agents on top of arbitrary data structures.

For these two reasons, our theory of agents will be closely coupled to a formal characterization of a body of software code that an agent $\mathfrak{a}$ may use. We denote the piece of software $\mathcal{S}^{\mathfrak{a}}$ associated with an agent $\mathfrak{a} \in \mathsf{A}$ by a triple $\mathcal{S} =_{def} (\mathcal{T}_{\mathcal{S}}, \mathcal{F}_{\mathcal{S}}, \mathcal{C}_{\mathcal{S}})$:

Definition 1 (Software Code $\mathcal{S} = (\mathcal{T}_{\mathcal{S}}, \mathcal{F}_{\mathcal{S}}, \mathcal{C}_{\mathcal{S}})$) *We may characterize the code on top of which an agent is built as a triple $\mathcal{S} =_{def} (\mathcal{T}_{\mathcal{S}}, \mathcal{F}_{\mathcal{S}}, \mathcal{C}_{\mathcal{S}})$ where:*

(1) $\mathcal{T}_{\mathcal{S}}$ is the set of all data types managed by $\mathcal{S}$,
(2) $\mathcal{F}_{\mathcal{S}}$ is a set of predefined functions which allow external processes to access (in principle) the data objects managed by the agent, and
(3) $\mathcal{C}_{\mathcal{S}}$ is a set of type composition operations. A type composition operator is a partial n-ary function c which takes as input, types $\tau_1, \ldots, \tau_n$ and yields as output, a type $c(\tau_1, \ldots, \tau_n)$. As c is a partial function, c may only be defined for certain arguments $\tau_1, \ldots, \tau_n$, i.e., c is not necessarily applicable on arbitrary types.

When an agent $\mathfrak{a}$'s identity is clear from context, we will often abuse notation and write $\mathcal{S}$ instead of $\mathcal{S}^{\mathfrak{a}}$. Intuitively, $\mathcal{T}_{\mathcal{S}}$ is the set of all data types that are managed by the agent. $\mathcal{F}_{\mathcal{S}}$ intuitively represents the set of all function calls supported by package $\mathcal{S}$'s application programmer interface (API). $\mathcal{C}_{\mathcal{S}}$ is the

set of ways of creating new data types from existing data types.

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 (13) and the *CORBA* framework (49) are existing industry standards consistent with this specification.

Message Box: The second important component of an agent's data structures is its message box. Every agent is required to have a message box data structure, together with some associated functions that read messages, send messages, etc. The precise set of such functions is described later on in Example 10 on page 13.

Belief Structures: Every agent maintains a specialized set of data structures called *belief structures*. These are the data structures in which the agent maintains its beliefs about the data other agents have, the way other agents reason, the actions other agents might take and/or the beliefs of beliefs of other agents about yet other agents. *An important aim of this paper is to define these structures, and to define what core functions are needed to manipulate these structures so that agents with different belief reasoning capabilities can be built.*

Meta-Agent Program: An agent program is a concept proposed in (21; 19). Agent programs encode the operating principles of an agent. This is very important because the developer of an agent should have the ability to write a transparent, declarative set of rules that control the actions that an agent takes. Given the fact that many of these actions physically modify existing data structures, the need to clearly enforce this is very important. Meta Agent Programs (maps for short) extend the concept of an agent program so that agents' operating principles may be influenced by their beliefs about themselves and/or 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 (8), and is not discussed in this paper.

Thus, at any given point in time, an agent $\mathfrak{a}$ *is managing a set of application specific data types, a message box data type, and a set of belief data structures to be specified in this paper.*

Definition 2 (State of an Agent) *At any given point t in time, the* state of an agent*, denoted* $\mathcal{O}_{\mathcal{S}}(t)$*, is the set of all objects the agent currently has—these types must be either in the base set of types in* $\mathcal{T}_{\mathcal{S}}$ *or must be a type obtained by composition using the composition operations in* $\mathcal{T}_{\mathcal{S}}$*.*

Throughout this paper, we assume that the only way two agents can communicate with one another is through messaging. There is no loss of gener-

ality in this assumption—if two agents share a common workspace, then this workspace can theoretically be viewed as part of the message boxes of the two agents.

Every agent therefore continuously executes the following cycle of operations—(i) read new messages, (ii) identify a set of actions that must be taken, and (iii) execute these actions concurrently. Thus, everytime the agent receives a message, its state changes, and the "operating principles" of the agent tell it (perhaps nondeterministically) what to do. The aim of this paper is to specify how a map determines a set of actions to take, given a "current" state of the agent, and given a new state change (i.e. receipt of a message).

Action Base: Every agent has an associated set of actions called its action base. Actions change the state of the agent and perhaps the state of other agents' `msgboxes`.

Integrity Constraints: Each agent has an associated set of integrity constraints—only states that satisfy these constraints are considered to be *valid* or *legal* states.

Action Constraints: 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 uses its *Meta-Agent Program* to determine what actions can be taken, what actions may not be taken, and what actions the agent must be taken. The map associated with an agent is a declarative specification of the agents' decision policies.

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 that an agent is built on top of. For the agent to perform logical reasoning on top of such third party data structures and code, the agent must have a language within which it can reason about the agent state. In this section, we introduce the concept of code calls and code call atoms introduced earlier in constraint form first in (36) (the form used here was first introduced in (1)). Code call atoms, and code call conditions defined in this section form the basic core language that allows an agent to reason about imperative data structures and code bases.

Given any type $\tau \in \mathcal{T}_\mathcal{S}$, we 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 $\mathtt{X.f_i}$

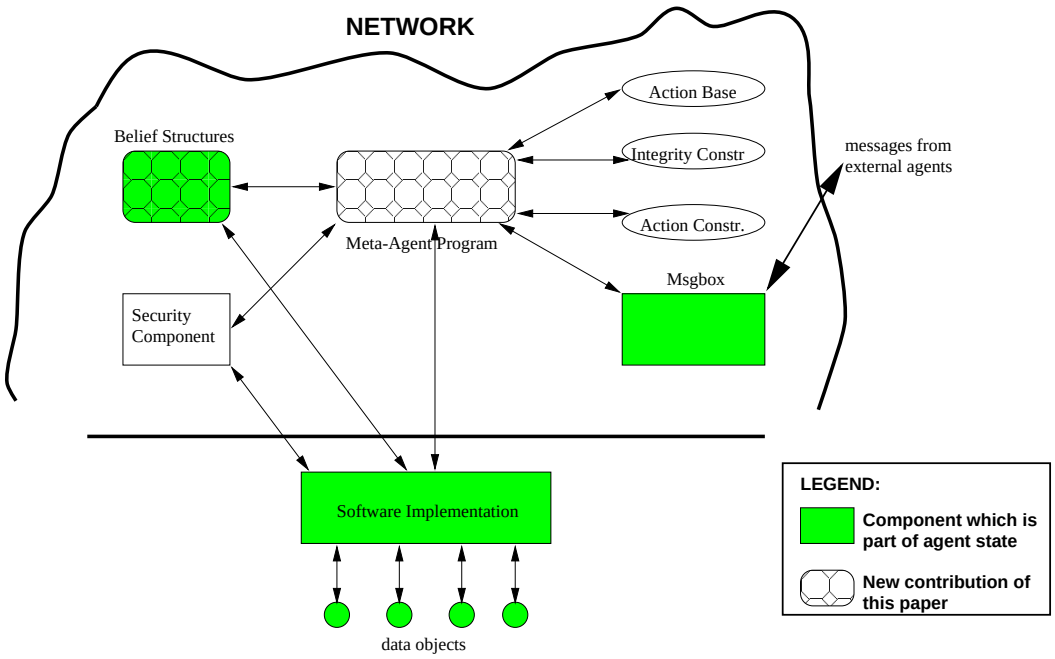

Figure 2. Overall Architecture

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 this case, we 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.

Definition 3 (Code Call cc) *Suppose* $\mathcal{S} =_{def} (\mathcal{T}_\mathcal{S}, \mathcal{F}_\mathcal{S}, \mathcal{C}_\mathcal{S})$ *is some software code and* $f \in \mathcal{F}_\mathcal{S}$ *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 a code call. *A code call is* ground *if all the* $\mathtt{d_i}$*'s are objects.*

It is important to note that an agent may provide access to several possible data sources/software packages. We will often abuse notation and write $\mathtt{a} : f(\mathtt{d_1}, \ldots, \mathtt{d_n})$ instead of $\mathcal{S} : f(\mathtt{d_1}, \ldots, \mathtt{d_n})$ when the software code that function f is defined in is clear from context.

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 produces. Here are some examples of code calls that are used in the RAMP example:

- heli : *setAltitude*(Alt) → Bool .
  This code call sets helicopter altitude to Alt. It returns **true** if it succeeds, otherwise it returns **false**.
- tank : *getPos*() → 2DPoint .
  This code call determines the current position of the tank.
- route : *flightPlan*(SourcePoint, DestPoint) → SequenceOf3DPoints .
  This code call creates a flight plan from point SourcePoint to point DestPoint. It returns the plan as a sequence of 3D points.
- track : *getTypeOfAgent*(AgentId) → (VehicleType, Prob) .
  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.
- coord : *findAttackTimeAndPos*(AgentId) → (Pos, Time, Route, Prob) .
  This code call tries to create a plan to attack the agent with id AgentId. The plan returned specifies a position (where the attack will take place) , a time point (when the attack will take place), a route to get there and the probability that the determination was correct.

Definition 4 (Code Call Atom in(X, cc)) *If* cc *is a code call, and* X *is either a variable symbol, or an object of the output type of* cc*, then* **in**(X, cc) *and* **not_in**(X, cc) *are* code call atoms.

It is important to note that **not_in**(X, cc) has absolutely nothing to do with negation as failure. **not_in**(X, cc) succeeds just in case X is not in the set of objects returned by the code call *cc*.

Definition 5 (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 s, t *are either variables or objects, then* $\mathtt{s} = \mathtt{t}$ *is a code call condition.*
(3) If s, 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 pos is the current position for the tank agent: **in**(pos, tank : *getPos*()). Here is a more complex code call condition.

in(P2, heli : *getPos*()) &
in(D, heli : *computeDistance*(P1, P2)) &
in(R, heli : *getMaxGunRange*()) &
D < R

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

Definition 6 (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* s, 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* Y *occurring in* $\mathtt{cc}_{\pi(i)}$, $root(\mathtt{Y})$ *is from the set* $\{root(\mathtt{X}_{\pi(\mathtt{j})}) \mid j < i\}$.

It is easily seen that the code call condition immediately preceding this definition is *not* safe. The reason for this is that the variable P1 is not instantiated by any of the **in** atoms. Had 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. In particular, if the current state of the agent satisfies these integrity constraints, then the agent may not execute actions that cause it to transition to a state where the integrity constraints are not satisfied.

Definition 7 (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:

$$\mathbf{in}(\mathtt{S}, \mathtt{tank}:\mathit{getSpeed}()) \;\Rightarrow\; \mathtt{S} < \mathtt{MaxSpeed}$$
$$\mathbf{in}(\mathtt{A}, \mathtt{heli}:\mathit{getAltitude}()) \;\Rightarrow\; \mathtt{A} < \mathtt{MaxAltitude}$$

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.

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.
- $\mathcal{P}$: an **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 introduce the concept of an action and describe how the effects of actions are implemented.

Definition 8 (Action; Action Atom) *An* action α *consists of six components:*

- ***Name:*** *A name, usually written* $\alpha(\mathtt{X}_1, \ldots, \mathtt{X}_n)$*, where the* $\mathtt{X}_i$*'s are root variables.*
- ***Schema:*** *A schema, usually written as* $(\tau_1, \ldots, \tau_n)$*, of types. Intuitively, this says that the variable* $\mathtt{X}_i$ *must be of type* τ_i*, for all* $1 \leq i \leq n$*.*
- ***Code:*** *This is a body of code that executes the action. Although this is a very important component, it is not needed for the declarative meaning of the action and therefore it is suppressed in the sequel.*
- ***Pre:*** *A code-call condition* χ*, called the* precondition *of the action, denoted by* $Pre(\alpha)$ *(*$Pre(\alpha)$ *must be* safe modulo the variables $\mathtt{X}_1$,...,$\mathtt{X}_n$*).*
- ***Add:*** *A set* $Add(\alpha)$ *of code-call conditions.*
- ***Del:*** *A set* $Del(\alpha)$ *of code-call conditions.*

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

The precondition $Pre(\alpha)$ *must be* safe modulo the variables $\mathtt{X}_1$,...,$\mathtt{X}_n$*. 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* $\mathtt{X}_1, \ldots, \mathtt{X}_n$ *and the root*

variables $Y_1, \ldots, Y_m$ *occurring in* $Pre(\alpha)$*, i.e., it is safe if every variable* Y *in* χ *such that* $root(Y) \in \{X_1 \ldots, X_n, Y_1, \ldots, Y_m\}$ *were considered as though it were a constant.*

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

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.

Example 9 (Some Actions of RAMP Agents) *We describe some actions of the Helicopter, Tank, Route and Coordination-Agents:*

Helicopter Agent:
Fly(From, To, Altitude, Speed)
Pre: **in**(From, heli : *getPos*())
Del: **in**(From, heli : *getPos*())
Add: **in**(To, heli : *getPos*())
Tank Agent:
drive(From, To, Speed)
Pre: **in**(From, tank : *getPos*())
Del: **in**(From, tank : *getPos*())
Add: **in**(To, tank : *getPos*())
Route Agent:
planRoute(Map, SourcePoint, DestPoint, VehicleType)
Pre: SourcePoint $\neq$ DestPoint
Del: {}
Add: **in**(**true**, route : *useMap*(Map)) &
in(Plan, route : *getPlan*(SourcePoint, DestPoint, VehicleType))
Coordination Agent:
attack(SetOfAgentIds, EnemyId)
Pre: SetOfAgentIds $\neq$ {}
Del: {}
Add: **in**(AP, coord : *coordinatedAttack*(SetOfAgentIds, EnemyId))

Example 10 (Message Box) *Throughout this paper, we 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 (52)) that can fill in (with incoming messages) or flush (when a message is physically sent off) this buffer. The msgbox operates on objects of the form* (i/o, "*src*", "*dest*", "*message*", "*time*")*. The parameter* i/o *signifies an incoming or outgoing message respectively. The variable* "*src*" *specifies the originator of the message whereas* "*dest*" *specifies the destina-*

tion. The "message" is a table consisting of triples of the form ("varName", "varType", "value") where "varName" is the name of the variable, "varType" is the type of the variable and the "value" is the value of the variable in string format. Finally, "time" denotes the time at which the message was sent.

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 (33) at a high level, and remote procedure calls at a much lower level). In order to provide maximal flexibility, we only 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 the form*

  $$sendMessage(\text{"}src\text{"}, \text{"}dest\text{"}, \text{"}message\text{"})$$

  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".*
- *getMessage(<src>): This causes a collection of quintuples*

  $$(\texttt{i}, \text{"}src\text{"}, \text{"}agent\text{"}, \text{"}msg\text{"}, \text{"}time\text{"})$$

  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* $=$.
- *getVar(<mssgId>, <varName>): This functions searches through all the triples in the "message" to find the requested variable. First, it converts the variable from the string format given by the "value" into its corresponding data type which is given by "varType". If the requested variable is not in the message determined by the "MssgId", then an error string is returned.*

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`.

Definition 11 (Action Base $\mathcal{AB}$) *An* action base, $\mathcal{AB}$, *is any finite collection of actions.*

In (21; 19), 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, $\texttt{conc_ex}(\texttt{Aset}, \texttt{Pre}, \texttt{Add}, \texttt{Del})$ 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.

Definition 12 (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.

Example 13 (Constraints for Fly and Attack) *Here are two simple constraints for the Fly and the Attack predicate:*

$\{$*fly_plane1*$(\texttt{X1}, \texttt{Y1}, \texttt{A1}, \texttt{S1})$, *fly_plane2*$(\texttt{X2}, \texttt{Y2}, \texttt{A2}, \texttt{S2})$ $\}$ $\hookleftarrow$ `Y1 =Y2`
$\{$*attack*$(\texttt{P})$ $\}$ $\hookleftarrow$ **in**$(\texttt{P}, \texttt{heli1} : getPos())$

The first says that plane1 and plane2 shall never fly to the same destination. The second says that `heli1` *shall never be attacked (this makes sense if the plane for which this constraint is valid knows that* `heli1` *is a friendly helicopter).*

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.

Definition 14 (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 Section 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 (39; 3). Moreover, the operator $\mathbf{Do}$ is in the spirit of the "praxiological" operator $\mathbf{E}_{\mathfrak{a}}A$ (30), which informally means that "agent $\mathfrak{a}$ sees to it that A is the case" (39, p.292).

Definition 15 (Action Program $\mathcal{P}$) *An* action rule *(*rule*, for short) is a clause* r *of the form*

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

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

An agent program $\mathcal{P}$ *is a finite collection of rules. An agent program* $\mathcal{P}$ *is* positive *if all its rules are positive.*

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 *allowedness* requirement on rules (35)).

4 Belief Language and Data Structures

In this section, we introduce the important notion of a *belief atom*. Belief atoms express the beliefs of one agent $\mathtt{a}$ about what holds in another agent's, say $\mathtt{b}$'s, state. They will be used later in Definition 28 on page 31 to define the notion of a *meta agent program*, which is central to this paper. When an agent $\mathtt{a}$ reasons about another agent $\mathtt{b}$, it must have some beliefs about $\mathtt{b}$'s underlying action base (*what actions can* $\mathtt{b}$ *take?*), $\mathtt{b}$'s agent program (*how will* $\mathtt{b}$ *reason?*) etc. These beliefs will be discussed later in more depth.

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

(1) We first describe in Section 4.1 a hierarchy of belief languages of increasing complexity as we go "up" the hierarchy. The designer of an agent choses an appropriate language from this hierarchy for her agent.
(2) We then define in Section 4.2 an intermediate structure called a *basic belief table*. Intuitively, a basic belief table maintained by agent $\mathtt{a}$ contains information about the beliefs $\mathtt{a}$ has about the states of other agents, as well as $\mathtt{a}$ itself. It also includes $\mathtt{a}$'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 (Section 4.3).
(4) We then extend (in Section 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* $\mathtt{a}$ *believes* ψ where ψ is a condition about some agent $\mathtt{b}$'s state, or about the actions that agent $\mathtt{b}$ might take.

It 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 on page 9: the more datatypes we have, the more types of code calls can be formulated in our language. We will introduce in Definition 35 on page 37 a precise notion of the set of *extended code calls*.

We are now ready to define 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

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

represent a belief of agent $\mathfrak{a}$ about what holds in the state of agent $\mathfrak{b}$. In this case, agent $\mathfrak{a}$ must have beliefs about agent $\mathfrak{b}$'s software package $\mathcal{S}^b$: the code call condition χ must be 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 34 on page 36).

From now on we refer to code call conditions satisfying the latter property (i.e. "χ has to be contained in $\mathcal{S}^b$") 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 action atoms 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: we simply assume they are already given.

Definition 16 (Belief Atom/Literal, $\mathcal{BA}t_1(\mathfrak{a},\mathfrak{b})$, $\mathcal{BL}it_1(\mathfrak{a},\mathsf{A})$) *Let $\mathfrak{a}, \mathfrak{b}$ be agents in A. Then we define the set $\mathcal{B}At_1(\mathfrak{a},\mathfrak{b})$ of $\mathfrak{a}$-belief atoms about $\mathfrak{b}$ of level 1 as follows:*

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

If $\mathcal{B}_{\mathfrak{a}}(\mathfrak{b},\chi)$ is a belief atom, then $\mathcal{B}_{\mathfrak{a}}(\mathfrak{b},\chi)$ and $\neg\mathcal{B}_{\mathfrak{a}}(\mathfrak{b},\chi)$ are called belief literals of level 1, *the corresponding set is denoted by $\mathcal{B}Lit_1(\mathfrak{a},\mathfrak{b})$.*

Let

$$\mathcal{B}At_1(\mathfrak{a},\mathsf{A}) =_{def} \bigcup_{b\in\mathsf{A}} \mathcal{B}At_1(\mathfrak{a},\mathfrak{b}) \quad \textit{and} \quad \mathcal{B}Lit_1(\mathfrak{a},\mathsf{A}) =_{def} \bigcup_{b\in\mathsf{A}} \mathcal{B}Lit_1(\mathfrak{a},\mathfrak{b})$$

be the set of all $\mathfrak{a}$-belief atoms (resp. belief literals) relative to A. This reflects the idea that agent $\mathfrak{a}$ 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:

Example 17 (Belief Atoms In RAMP)

- $\mathcal{B}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : \mathit{getPos}()))$
  This belief atom says that the agent, heli1 *believes that agent* tank1*'s current state indicates that* tank1*'s current position is* pos1.
- $\mathcal{B}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{F}\mathit{attack}(\mathtt{pos1}, \mathtt{pos2}))$
  This belief atom says that the agent, heli1 *believes that agent* tank1*'s current state indicates that it is forbidden for* tank1 *to attack from* pos1 *to* pos2.
- $\mathcal{B}_{\mathtt{heli3}}(\mathtt{tank1}, \mathbf{O}\mathit{drive}(\mathtt{pos1}, \mathtt{pos2}, 35))$
  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 $\mathcal{BL}it_1(\mathtt{a}, \mathtt{A})$ does not allow agent $\mathtt{a}$ to have beliefs of the form *"Agent* $\mathtt{b}$ *believes that agent* $\mathtt{c}$*'s state contains code call condition* χ*"*, i.e. agent $\mathtt{a}$ cannot express beliefs it has about the beliefs of another agent.

The definition below introduces nested beliefs and also a general *belief language*. We introduce the following notation: for a given set X of formulae we denote by $\mathbf{Conj}_\neg(X)$ the set of all conjunctions consisting of elements of X or their negations: $x_1 \wedge \neg x_2 \wedge \ldots \wedge x_n$, where $x_i \in X$. We emphasize that this does not correspond to the usual closure of X under & and $\neg$: in particular, it does not allow us to formulate disjunctions, if X consists of atoms.

Definition 18 (Nested Beliefs $\mathcal{BL}it_i(\mathtt{a}, \mathtt{b})$, Belief language $\mathcal{BL}_i^{\mathtt{a}}$) *In the following let* $\mathtt{a}, \mathtt{b} \in \mathtt{A}$*. We want to define* $\mathcal{BL}_i^{\mathtt{a}}$*, the* belief language of agent $\mathtt{a}$ of level i*. This is done recursively as follows.*

i $\leq$ 1: *In accordance with Definition 16 (where we already defined* $\mathcal{B}At_1(\mathtt{a}, \mathtt{b})$*) we denote by* $\mathcal{B}At_0(\mathtt{a}, \mathtt{b})$ *as well as by* $\mathcal{BL}it_0(\mathtt{a}, \mathtt{b})$

$$\{\phi \mid \phi \text{ is a compatible code call condition or action atom}\}$$

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

$$\mathcal{BL}_0(\mathtt{a}, \mathtt{b}) =_{def} \mathcal{B}At_0(\mathtt{a}, \mathtt{b})$$
$$\mathcal{BL}_1(\mathtt{a}, \mathtt{b}) =_{def} \mathbf{Conj}_\neg(\mathcal{B}At_1(\mathtt{a}, \mathtt{b})),$$

i.e. the set of formulae $\mathcal{BA}t_0(\mathfrak{a},\mathfrak{b})$*, resp. the of all conjunctions of belief literals from* $\mathcal{BA}t_1(\mathfrak{a},\mathfrak{b})$.

We call

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

$$\mathcal{BL}_1^{\mathfrak{a}} =_{def} \mathbf{Conj}_{\neg}(\bigcup_{\mathfrak{b}\in\mathsf{A}} \mathcal{BL}_1(\mathfrak{a},\mathfrak{b}))$$

the belief languages of agent $\mathfrak{a}$ of level 0, resp. of level 1.

$\mathsf{i} > \mathbf{1}$**:** *To define nested belief literals we set for* $i > 1$

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

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

This finishes the recursive definition of $\mathcal{BL}it_i(\mathfrak{a},\mathfrak{b})$.

The definition of the belief language of agent $\mathfrak{a}$ of level i *can now be given directly:*

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

where

$$\mathcal{BL}_i(\mathfrak{a},\mathfrak{b}) =_{def} \mathbf{Conj}_{\neg}(\mathcal{BA}t_i(\mathfrak{a},\mathfrak{b})).$$

Finally we define the maximal belief language an agent $\mathfrak{a}$ *can have:*

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

Formulae in this language are also called general belief formulae.

We will later also use the following definitions:

(1) $\mathcal{BA}t_i(\mathfrak{a},\mathsf{A}) =_{def} \bigcup_{\mathfrak{b}\in\mathsf{A}} \mathcal{BA}t_i(\mathfrak{a},\mathfrak{b})$ *is called the set of belief atoms of depth* i.

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

(3) We define

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

At first sight the above definition looks overly complicated. The reason is that every agent keeps track of only its *own* beliefs, not those of other agents (we will see later in Lemma 24 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 $\{\&, \neg\}$ 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 also use 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 is not necessary. We discuss this point again in Lemma 24.

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

Example 19 (Belief Formulae for RAMP) *The following are belief formulae from* $\mathcal{BL}_1^{\mathtt{heli1}}$, $\mathcal{BL}_2^{\mathtt{tank1}}$ *and* $\mathcal{BL}_3^{\mathtt{coord}}$.

- $\boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : \mathit{getPos}()))$.
  This formula is in $\mathcal{BL}_1^{\mathtt{heli1}}$. *It says that agent* heli1 *believes that agent* tank1*'s current state indicates that* tank1*'s current position is* pos1.
- $\boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli1}, \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : \mathit{getPos}())))$.
  This formula is in $\mathcal{BL}_2^{\mathtt{tank1}}$. *It says that agent* tank1 *believes that agent* heli1 *believes that agent* tank1*'s current position is* pos1.
- $\boldsymbol{\mathcal{B}}_{\mathtt{coord}}(\mathtt{tank1}, \boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli1}, \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank2}, \mathbf{in}(\mathtt{pos2}, \mathtt{tank2} : \mathit{getPos}()))))$.
  This formula is in $\mathcal{BL}_3^{\mathtt{coord}}$. *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 of the above belief languages:

$$\boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli1}, \boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank} : \mathit{getPos}()))).$$

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.

Definition 20 (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 $\mathsf{h} \in \mathsf{A}$ *and* $\phi \in \mathcal{BL}_i^{\mathsf{h}}$, $i \in \mathbb{N}$.

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 $\phi \in \mathcal{BL}_1^{\mathsf{b}}$.

Example 21 (Basic Belief Table for RAMP Agents) *We define suitable basic belief tables for agent* tank1 *(Table 1) and* heli1 *(Table 2).*

Agent	**Formula**
heli1	$\mathbf{in}(\mathsf{pos1}, \mathsf{heli1} : \mathit{getPos}())$
heli2	$\boldsymbol{\mathcal{B}}_{\mathsf{heli2}}(\mathsf{tank1}, \mathbf{in}(\mathsf{pos1}, \mathsf{tank1} : \mathit{getPos}()))$
tank2	$\boldsymbol{\mathcal{B}}_{\mathsf{tank2}}(\mathsf{heli1}, \boldsymbol{\mathcal{B}}_{\mathsf{heli1}}(\mathsf{tank1}, \mathbf{in}(\mathsf{pos3}, \mathsf{tank1} : \mathit{getPos}())))$

Table 1
A Basic Belief Table for agent tank1.

Agent	**Formula**
heli2	$\mathbf{in}(\mathsf{pos2}, \mathsf{heli2} : \mathit{getPos}())$
tank1	$\mathbf{in}(\mathsf{pos1}, \mathsf{tank1} : \mathit{getPos}())$
tank1	$\boldsymbol{\mathcal{B}}_{\mathsf{tank1}}(\mathsf{heli1}, \mathbf{in}(\mathsf{pos1}, \mathsf{heli1} : \mathit{getPos}()))$
tank2	$\boldsymbol{\mathcal{B}}_{\mathsf{tank2}}(\mathsf{tank1}, \boldsymbol{\mathcal{B}}_{\mathsf{tank1}}(\mathsf{heli1}, \mathbf{in}(\mathsf{pos4}, \mathsf{heli1} : \mathit{getPos}())))$

Table 2
A Basic Belief Table for agent heli1.

These tables indicate that tank1 *and* heli1 *work closely together and know their positions. Both believe that the other knows about both positions.* tank1 *also believes that* tank2 *believes that in* heli2*'s state,* tank1 *is in position* pos3 *(which is actually wrong).*

heli1 *thinks that* tank2 *believes that* tank1 *believes that* heli1 *is in position* 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 expressiveness and utility of our overall framework. For example the background theory may contain some epistemic axioms about beliefs, some inference rules and the relation between belief formulae could be an entailment relation with respect to the chosen background theory (see also (9; 10)).

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

Definition 22 (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 $\mathfrak{h} \in \mathsf{A}$, $\mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}}$ *is a belief semantics over* $\mathcal{BL}^{\mathfrak{h}}_i$ *and* $i \in \mathbb{N}$ *is fixed. In addition we require at most one entry per agent* $\mathfrak{h}$. *Hence,* $\mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}}$ *determines an entailment relation*

$$\phi \mid\!\!=_{\mathcal{B}Sem^{\mathfrak{a}}_{\mathfrak{h}}} \psi$$

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 35 on page 37) over* $\mathbf{BSemT}^{\mathfrak{a}}$*:*

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

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

Example 23 (Belief Semantics Tables for RAMP Agents) *We briefly describe what suitable Belief Semantics Table for* heli1 *and* tank1 *may look like. We have to define entailment relations* $\mathcal{B}Sem^{\mathtt{tank1}}_{\mathtt{tank2}}$, $\mathcal{B}Sem^{\mathtt{tank1}}_{\mathtt{heli1}}$, $\mathcal{B}Sem^{\mathtt{tank1}}_{\mathtt{heli2}}$, *and* $\mathcal{B}Sem^{\mathtt{heli1}}_{\mathtt{tank1}}$, $\mathcal{B}Sem^{\mathtt{heli1}}_{\mathtt{tank2}}$, $\mathcal{B}Sem^{\mathtt{heli1}}_{\mathtt{heli2}}$. *For simplicity we restrict these entailment relations to belief formulae of level at most* 1, *i.e.* $\mathcal{BL}^{\mathfrak{h}}_1$.

(1) $\mathcal{B}Sem^{\mathtt{heli1}}_{\mathtt{tank1}}$*: The smallest entailment relation satisfying the schema*

$$\boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{tank1.1}, \chi) \rightarrow \chi.$$

This says that heli1 *believes that all beliefs of* tank1 *about* tank1.1 *are actually true:* tank1 *knows all about* tank1.1.

(2) $\mathcal{B}Sem^{\mathtt{heli1}}_{\mathtt{tank2}}$*: The smallest entailment relation satisfying the schema*

$$\boldsymbol{\mathcal{B}}_{\mathtt{tank2}}(\mathtt{tank2.1}, \chi) \rightarrow \chi.$$

This says that heli1 *believes that all beliefs of* tank2 *about* tank2.1 *are actually true:* tank2 *knows all about* tank2.1.

(3) $\mathcal{B}Sem^{\mathtt{tank1}}_{\mathtt{heli1}}$*: The smallest entailment relation satisfying the schema*

$$\boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1}, \chi) \rightarrow \chi.$$

This says that tank1 *believes that if* heli1 *believes in* χ *for* tank1*, then this is true (*heli1 *knows all about* tank1*. A particular interesting instance of* χ *is* $\mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : getPos())$.

(4) $\mathcal{B}Sem^{\mathtt{tank1}}_{\mathtt{heli2}}$*: The smallest entailment relation satisfying the schema*

$$\boldsymbol{\mathcal{B}}_{\mathtt{heli2}}(\mathtt{tank2}, \chi) \wedge \boldsymbol{\mathcal{B}}_{\mathtt{heli2}}(\mathtt{tank2.1}, \chi) \rightarrow \chi.$$

This says that tank1 *believes that if* heli2 *believes that* χ *is true both for* tank2 *and* 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 $\mathcal{BL}^{\mathsf{h}}_i \times \mathcal{BL}^{\mathsf{h}}_i$. We briefly illustrate *(1) which sort of semantics can be expressed* and *(2) how our framework can be suitably restricted for practical applications.*

The following two simple axioms that can be built into a semantics show the generality and flexibility of our framework:

$$(1)\ \boldsymbol{\mathcal{B}}_{\mathsf{h}_2}(\mathsf{h}, \chi) \Rightarrow \boldsymbol{\mathcal{B}}_{\mathsf{h}_2}(\mathsf{h}', \chi)$$

$$(2)\ \boldsymbol{\mathcal{B}}_{\mathsf{h}_2}(\mathsf{h}, \chi) \Rightarrow \qquad \chi$$

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 18. 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 (21; 19) and the various semantics Sem for agent programs (i.e. with no belief atoms), we now show how such a semantics Sem naturally induces a semantics $\mathcal{B}Sem^{\mathsf{a}}_{\mathsf{h}}$ for use in a belief semantics table. These semantics can be implemented and handled as built-ins. Entries in $\mathbf{BSemT}^{\mathsf{a}}$ then have the form

$$\langle \mathsf{h}_1, \mathsf{Sem}_{feas} \rangle$$

$$\langle \mathsf{h}_2, \mathsf{Sem}_{rat} \rangle$$

$$\langle \mathsf{h}_3, \mathsf{Sem}_{reas} \rangle$$

meaning that agent a believes that the agents h_i behave according to the indicated semantics—such semantics are well understood for agent programs without beliefs.

The idea is to use the semantics Sem of the agent program $\mathcal{P}^{\mathsf{a}}(\mathsf{b})$ (that a believes b to have) for the evaluation of the belief formulae. However, this is 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 depends on (1) an agent 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 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 $\langle \mathsf{h}_1, \mathsf{Op}\, \alpha(\vec{t}) \rangle$ is a belief of a about the actions of h_1 holds. Consequently action atoms can be seen as conditions on h_1's agent program.

We show below how to define belief semantics on belief languages of levels 0 and 1. Recall that belief formulae contain *both* code call conditions and action atoms which arw evaluated over different domains. Therefore for a formula ϕ which is a *conjunction* of code call conditions (ccc's for short) 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 atoms occuring in ϕ.

The remark below specifies a *methodology* for incorporating agent semantics into our framework.

Remark 24 (Sem for Agent Programs induces $\mathcal{BS}em^{\mathsf{a}}_{\mathsf{h}}$) *Let* Sem *be the* reasonable, rational *or* feasible *semantics for agent programs (i.e. not containing beliefs). Suppose agent* a *believes that agent* h *reasons according to* Sem. *Let* $\mathcal{P}(\mathsf{h})$ *be the agent program of* h *and* $\mathcal{O}(\mathsf{h})$, $\mathcal{AC}(\mathsf{h})$ *and* $\mathcal{IC}(\mathsf{h})$ *the state, action constraints and integrity constraints of* h. *Then there is a basic belief table* $\mathbf{BSemT}^{\mathsf{a}}$ *and a belief semantics* $\mathcal{B}Sem^{\mathsf{a}}_{\mathsf{h}}$ *induced by* Sem *such that*

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

More generally: let $i \in \mathbb{N}$ *and suppose agent* a *believes that agent* h_1 *believes that agent* h_2 *believes that ... believes that agent* h_{i-1} *acts according to* $\mathcal{P}^{\mathsf{a}}(\sigma)$ *(where* $\sigma =_{def} [\mathsf{h}_1, \mathsf{h}_2, \ldots, \mathsf{h}_{i-1}]$*) and state* $\mathcal{O}(\sigma)$ [1] *. Then there is a basic belief table* $\mathbf{BSemT}^{\mathsf{a}}$ *and a belief semantics* $\mathcal{B}Sem^{\mathsf{a}}_{\sigma}$ *induced by* Sem *on a suitably restricted subset of* $\mathcal{BL}^{\mathsf{h}}_1 \times \mathcal{BL}^{\mathsf{h}}_1$ *such that*

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

[1] See Definition 34 and Definition 31 for a detailed introduction of these concepts.

The reason that the above statement is not amenable to a proof or disproof is the part "*... agent* a *believes that agent* h *reasons according to* Sem ". It is intuitely clear what this means, but the precise meaning is given by argument below.

Justification of Remark. We define a belief semantics $\mathcal{B}Sem^{\mathsf{a}}_{\mathsf{h}}$ on $\mathcal{BL}^{\mathsf{h}}_0 \times \mathcal{BL}^{\mathsf{h}}_0$ with respect to a state $\mathcal{O}$, $\mathcal{AC}$, and $\mathcal{IC}$ as follows:

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

We now define a belief semantics $\mathcal{B}Sem^{\mathsf{a}}_{\mathsf{h}}$ on $\mathcal{BL}^{\mathsf{h}}_1 \times \mathcal{BL}^{\mathsf{h}}_1$ 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: $\{\boldsymbol{\mathcal{B}}_{\mathsf{h}}(\mathsf{c}, \phi) \mid \ \phi \text{ is a code call condition or an action atom}\}$.
(3) For a belief formula ϕ of level 1 which has the form $\boldsymbol{\mathcal{B}}_{\mathsf{h}}(\mathsf{c}, \phi_1) \wedge \cdots \wedge \boldsymbol{\mathcal{B}}_{\mathsf{h}}(\mathsf{c}, \phi_n)$ we let

$$\mathrm{CCC}(\phi) =_{def} \mathrm{CCC}(\phi_1) \wedge \cdots \wedge \mathrm{CCC}(\phi_n)$$

and

$$\mathrm{ACT}(\phi) =_{def} \mathrm{ACT}(\phi_1) \wedge \cdots \wedge \mathrm{ACT}(\phi_n).$$

(4) We define:

$$\phi \models_{\mathcal{B}Sem^{\mathsf{a}}_{\mathsf{h}}} \psi \text{ by } \begin{cases} 1.\ \mathrm{ACT}(\psi) \in \mathsf{Sem}_{\mathsf{c}}(\mathcal{P}^{\mathsf{a}}([\mathsf{h}, \mathsf{c}]) \cup \{\mathrm{ACT}(\phi)\}) \\ \quad \text{wrt. } \mathcal{O} \cup \mathrm{CCC}(\phi). \\ 2.\ \mathcal{O} \cup \mathrm{CCC}(\phi) \models \mathrm{CCC}(\psi). \\ 3.\ \mathcal{AC} \text{ are satisfied wrt. } \mathcal{P}^{\mathsf{a}}([\mathsf{h}, \mathsf{c}]) \cup \{\mathrm{ACT}(\phi)\}. \\ 4.\ \mathcal{O} \cup \mathrm{CCC}(\phi) \models \mathcal{IC} \end{cases}$$

The notation $\mathcal{P}^{\mathsf{a}}([\mathsf{h}, \mathsf{c}])$ denotes the program that a believes h to believe about c. The sequences σ will be introduced in Definition 31 on page 34.

4.4 Belief Tables

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

Definition 25 (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 26 on page 29).*

The projection of $\mathbf{BT}^{\mathfrak{a}}$ *on the first two entries coincides with the basic belief table introduced in Definition 20 on page 22. Thus, every belief table induces a (possibly empty) basic belief table.*

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

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

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* $\mathcal{B}Sem_{\mathfrak{h}}^{\mathfrak{a}}$*.*

The reader may well ask about whether belief update functions should also be included. In fact, belief updating is an *action* that may be included in an agent, should the agent developer deem it necessary for his agent. Such an action can be implemented using any standard belief updating algorithm proposed in the literature.

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.

Why do we require that $\mathbf{BT}^{\mathtt{a}}$: proj-select(agent, =, $\mathtt{h}$) selects only those triples where the third entry set to **true**? The reason is that when we evaluate the belief condition (to ensure compatibility in Definition 25) we do this recursively and thus end up with triples having the third entry **true**. As we could, of course, have added another argument to the function which allows us to specify the third entry.

In the preceding definition we introduced the notion of a belief table but we have not specified its third entry, namely the *belief condition*. Intuitively, $\langle \mathtt{b}, \phi, \chi_{\mathcal{B}} \rangle$ means that

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

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

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

Note 1 *There is considerable work on modal multiagent logics where agent* $\mathtt{a}$ *has beliefs about agent* $\mathtt{b}$ *and so on. In general, in such logics, each agent has an associated belief theory about another agent. This is consistent with our view. Given an agent* $\mathtt{b}$*, agent* $\mathtt{a}$*'s base-beliefs about* $\mathtt{b}$ *consists of the set of all formulas* ϕ *such that there is a row* r *in* $\mathbf{BT}^{\mathtt{a}}$ *with* $\mathtt{r.Agent} = \mathtt{b}$ *and* $\mathtt{r.Condition} =$ **true** *and* $\mathtt{r.Formula} = \phi$*. Agent* $\mathtt{a}$*'s total set of beliefs about agent* $\mathtt{b}$ *is the set of all formulas derivable from its base beliefs using the semantics that agent* $\mathtt{a}$ *believes agent* $\mathtt{b}$ *is using—which is contained in* $\mathbf{BSemT}^{\mathtt{a}}$*.*

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

Definition 26 (Belief Conditions $\mathcal{BC}ond^{\mathtt{a}}(\mathtt{h})$) *The set* $\mathcal{B}Cond^{\mathtt{a}}(\mathtt{h})$ *of belief conditions of agent* $\mathtt{a}$ *is defined inductively as follows:*

(1) Every code call condition χ *of agent* $\mathtt{a}$ *compatible with agent* $\mathtt{h}$ *is in* $\mathcal{B}Cond^{\mathtt{a}}(\mathtt{h})$*.*

(2) If $\mathtt{X}$ *is an entry in the basic belief table (or, equivalently the projection of an entry of the belief table* $\mathbf{BT}^{\mathtt{a}}$ *on the first two arguments) or a variable over basic belief table tuples, then*

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

is in $\mathcal{B}Cond^{\mathtt{a}}(\mathtt{h})$*.*

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

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

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

As belief conditions corresponding to step (1) *above will be checked in what agent* $\mathtt{a}$ *believes agent* $\mathtt{b}$*'s state to be, we introduce the following notation:*

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

Note that $\mathtt{h}_part(\chi)$ *consists of conditions that have to be checked in what* $\mathtt{a}$ *believes agent* $\mathtt{h}$*'s state is, while* $\mathtt{a}_part(\chi)$ *refers to the belief tables of agent* $\mathtt{a}$. *Similarly, for other agent* $\mathtt{b}$, $\mathtt{c}$, $\mathtt{d}$ *we have the notion of* $\mathtt{b}_part(\chi)$, $\mathtt{c}_part(\chi)$ *and* $\mathtt{d}_part(\chi)$.

Agent	**Formula**	**Condition**
heli1	$\mathbf{in}(\mathtt{pos1}, \mathtt{heli1} : getPos())$	**true**
heli2	$\mathcal{B}_{\mathtt{heli2}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : getPos()))$	$\mathcal{B}cond_1^{\mathtt{tank1}}$
tank2	$\mathcal{B}_{\mathtt{tank2}}(\mathtt{heli1}, \mathcal{B}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos3}, \mathtt{tank1} : getPos())))$	$\mathcal{B}cond_2^{\mathtt{tank1}}$

Table 3
A Belief Table for agent tank1.

Example 27 (Belief Table for RAMP Agents Revisited) *We now consider Table 3 on the next page and extend our basic belief tables for agent* tank1 *(Table 1 on page 22) and* heli1 *(Table 2 on page 22). Let* $\mathcal{B}cond_1^{\mathtt{tank1}}$ *be the code call atom* $\mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : getPos())$ *and define* $\mathcal{B}cond_2^{\mathtt{tank1}}$ *by*

$$\mathbf{in}(\langle \mathtt{heli1}, \textit{belief atom} \rangle, \mathbf{BT}^{\mathtt{a}} : \text{proj-select}(\text{agent}, =, \mathtt{heli1})),$$

where

$$\textit{belief atom} =_{def} \mathcal{B}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos3}, \mathtt{tank1} : getPos())).$$

The first row in the table says that tank1 *unconditionally believes that in* heli1*'s state, the position for* heli1 *is* pos1.

The second row in the belief table above, says that tank1 *believes that if* tank1*'s position is* pos1, heli2 *believes that in* tank1*'s state the position of* tank1 *is* pos1.

The third row in the belief table says that if tank1 *believes* heli1 *believes that* tank1*'s position is* pos3*, then* tank2 *believes* heli1 *believes* tank1*'s position is* pos3.

The table for heli1 *is as shown in Table 4, where* $\mathcal{B}cond_1^{\mathtt{heli1}}$ *stands for*

$$\mathbf{in}(\mathtt{pos2}, \mathtt{heli2} : getPos())$$

and $\mathcal{B}cond_2^{\mathtt{tank1}}$ *is defined by*

$$\mathbf{in}(\langle \mathtt{tank1}, \textit{belief atom} \rangle, \mathbf{BT}^{\mathtt{a}} : \text{proj-select}(\text{agent}, =, \mathtt{tank1})),$$

where

$$\textit{belief atom} =_{def} \boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli1}, \mathbf{in}(\mathtt{pos4}, \mathtt{heli1} : getPos())).$$

Agent	**Formula**	**Condition**
heli2	$\mathbf{in}(\mathtt{pos2}, \mathtt{heli2} : getPos())$	**true**
tank1	$\mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : getPos())$	**true**
tank1	$\boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli1}, \mathbf{in}(\mathtt{pos1}, \mathtt{heli1} : getPos()))$	$\mathcal{B}cond_1^{\mathtt{heli1}}$
tank2	$\boldsymbol{\mathcal{B}}_{\mathtt{tank2}}(\mathtt{tank1}, \boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli1}, \mathbf{in}(\mathtt{pos4}, \mathtt{heli1} : getPos())))$	$\mathcal{B}cond_2^{\mathtt{heli1}}$

Table 4
A Belief Table for agent heli1.

We are now in a position to formally define a meta agent program, i.e. a program which formalizes the actions and the circumstances under which an agent $\mathtt{a}$ will execute these actions based not only on its own state but also on its beliefs about other agent's states, and actions those other agents might take.

Definition 28 (Meta Agent Program (map) $\mathcal{BP}$) *A* meta agent rule*, (*mar *for short), for agent* $\mathtt{a}$ *is an expression* 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* $\mathcal{BL}it_\infty(\mathtt{a}, \mathtt{A})$.

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

Note that belief atoms are not allowed in the head of rules. Thus a meta agent program does not define beliefs: it only uses them to derive action status atoms. Beliefs are solely defined by the belief data structures (belief table and through the belief semantics table).

Example 29 (map's for RAMP-Agents) *Let* heli1*'s meta agent program be as follows:*

$$\begin{array}{lll} \mathbf{P}\ \mathit{attack}(\mathtt{P1},\mathtt{P2}) & \leftarrow & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1},\mathbf{in}(\mathtt{P2},\mathtt{tank1}:\mathit{getPos}()))\ , \\ & & \mathbf{P}\ \mathit{fly}(\mathtt{P1},\mathtt{P3},\mathtt{A},\mathtt{S})\,, \\ & & \mathbf{P}\ \mathit{attack}(\mathtt{P3},\mathtt{P2})\,. \end{array}$$

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:*

$$\begin{array}{lll} \mathbf{O}\ \mathit{attack}(\mathtt{P1},\mathtt{P2}) & \leftarrow & \mathbf{O}\ \mathit{driveRoute}([\mathtt{P0},\mathtt{P1},\mathtt{P2},\mathtt{P3}],\mathtt{S})\,, \\ & & \boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{tank2},\mathbf{in}(\mathtt{P2},\mathtt{tank2}:\mathit{getPos}()))\,. \end{array}$$

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 $\mathcal{S}^{\mathfrak{a}} = (\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}, \mathcal{F}_{\mathcal{S}}^{\mathfrak{a}})$ 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

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

Before concluding this section, we briefly discuss the possibility of beliefs being incomplete. Agent $\mathfrak{a}$ represents its beliefs about another agent $\mathfrak{b}$ via its associated belief table and belief semantics table. However, it is entirely possible that one or both of these tables in incomplete, incorrect, or both. At a fixed instant of time t all agents (including human beings!) can only reason about agent $\mathfrak{b}$ based on their beliefs about $\mathfrak{b}$ and their background knowledge (i.e. their state). Agent $\mathfrak{a}$ may or may not (depending upon the application) need to complete its beliefs about agent $\mathfrak{b}$. In the case of the RAMP example, a friendly vehicle agent $\mathfrak{a}$ can only guess what an enemy agent $\mathfrak{b}$ will do. Such guesses are likely to be wrong (as is the case with all guesses). However, agent $\mathfrak{a}$'s associated map may trigger actions to be taken that reduce the ignorance about agent $\mathfrak{b}$. For example, it could send a message (which is an action!) to another battlefield analysis agent, requesting a list of all possible scenarios describing the intentions of this action, together with plausibility rankings of the scenarios, as well as the dangers and opportunities posed by each scenario.

It can then make its decisions based on that. Another action it could possibly take is to bomb part of the route of enemy vehicle b to remove uncertainty. For example, if agent b us moving north on a road that forks to the left towards friendly asset A_1 and forks to the right towards friendly asset A_2, by destroying or mining the fork to A_2, it can effectively reduce uncertainty about the intent of agent b—in this case, even though it still doesn't know what agent b's intentions are, it can turn the situation to its advantage. All these types of reasoning can be encoded as rules within the map paradigm.

We did not, however, incorporate a Kripke-style semantics as a built-in feature of our framework (as described in (25)). Such semantics can be plugged in, if needed, but the designer of the agent has the freedom (or the burden) to do so. Note that we are describing a framework suitable for arbitrary software agents, not for a particular *knowledge and belief* agent.

5 Meta-Agent Programs: Semantics

It remains to define the *semantics* of meta agent programs. As in the case of agent programs without any metaknowledge (summarized briefly in the appendix), 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.

Definition 30 (Belief Status Set $\mathcal{BS}$) *A belief status set $\mathcal{BS}$ of agent a, also written $\mathcal{BS}(\mathsf{a})$, is a set consisting of two kinds of elements:*

- *ground action status atoms over $\mathcal{S}^{\mathsf{a}}$ and*
- *belief literals from $\mathcal{BA}t_{\infty}(\mathsf{a}, \mathsf{A})$ 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 60).

We emphasize the different treatment of status *atoms* and belief *literals*. A belief status set is considered to be

- *complete* with respect to action status atoms, i.e. the absence of a particular status atom means its negation is true,
- *incomplete* with respect to belief literals, i.e. the absence of $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b}, \chi)$ does not mean that we assume $\neg\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b}, \chi)$.

Thus while negation signs in front of belief atoms are not handled as negation-as-failure (they are completely determined by the belief—and the belief—semantics table) the negation in front of status atoms is given a nonmonotonic interpretation (see below).

The handling of incomplete beliefs through the belief tables may also lead to inconsistencies: if the belief table contains both $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b}, \chi)$ and $\neg\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b}, \chi)$, any associated belief status set $\mathcal{BS}$ will also contain those literals. Together with the belief semantics compatibility condition (see Definition 46) this will result in a belief status set containing the whole language of belief literals. This can only be avoided by imposing that the belief table and the belief semantics table are consistent. To ensure this, however, is the responsibility of the agent designer.

To determine whether a belief status set is feasible, we must examine its compliance with

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

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

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

Definition 31 (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 $\mathsf{a} \in \mathsf{A}$ and $[\rho]$ is a sequence, then $[\mathsf{a}]$,$[\mathsf{a}, \rho]$, $[\rho, \mathsf{a}]$ are sequences.

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

The overall intuition of the formula $\boldsymbol{\mathcal{B}}_{\mathsf{a}}(\mathsf{b}, \boldsymbol{\mathcal{B}}_{\mathsf{b}}(\mathsf{c}, \boldsymbol{\mathcal{B}}_{\mathsf{c}}(\mathsf{d}, \chi)))$ is that if we keep agent a in mind, then agent a believes in a code call condition of type $[\mathsf{b}, \mathsf{c}, \mathsf{d}]$, i.e. a ccc that b believes that c believes that it holds in 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 $[\mathfrak{a}, \mathfrak{b}, \ldots] = \sigma$.

Definition 32 (Induced Status Set $\Pi_{\mathfrak{b}}^{\mathbf{action}}(\mathcal{BS})$ and State $\Pi_{\mathfrak{b}}^{\mathbf{state}}(\mathcal{BS})$) *Let $\mathfrak{a}, \mathfrak{b}$ be agents and $\mathcal{BP}$ a map used by agent $\mathfrak{a}$. Every belief status set $\mathcal{BS}$ of agent $\mathfrak{a}$ induces the following two sets describing $\mathfrak{a}$'s beliefs about $\mathfrak{b}$'s actions and $\mathfrak{b}$'s state*

$$\Pi_{\mathfrak{b}}^{\text{action}}(\mathcal{BS}) =_{def} \{\ \mathit{Op}\,\alpha(\vec{t}) \mid \boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathit{Op}\,\alpha(\vec{t})) \in \mathcal{BS},\ \textit{where } \mathit{Op} \in \{\mathbf{O}, \mathbf{W}, \mathbf{P}, \mathbf{F}, \mathbf{Do}\}\}$$
$$\Pi_{\mathfrak{b}}^{\text{state}}(\mathcal{BS}) =_{def} \{\quad \chi \quad \mid \boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \chi) \in \mathcal{BS} \ \textit{and}\ \chi \ \textit{is a code call condition}\}$$

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

Analogously, for any sequence σ, $\mathcal{BS}$ induces the following two sets:

$\Pi_{\sigma}^{\text{action}}(\mathcal{BS})$ *describing $\mathfrak{a}$'s belief about actions corresponding to σ*

$\Pi_{\sigma}^{\text{state}}(\mathcal{BS})$ *describing $\mathfrak{a}$'s belief about the state corresponding to σ,*

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

The formal definition of $\Pi_{\sigma}^{\text{action}}(\mathcal{BS})$ and $\Pi_{\sigma}^{\text{state}}(\mathcal{BS})$ is by induction on the structure of σ. As it should be clear, we avoid this technical definition. Instead we shortly illustrate the case for $\sigma = [\mathfrak{b}, \mathfrak{c}]$. Then

$$\Pi_{[\mathfrak{b},\mathfrak{c}]}^{\text{action}}(\mathcal{BS}) =_{def} \{\ \mathsf{Op}\,\alpha(\vec{t}) \mid \boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \boldsymbol{\mathcal{B}}_{\mathfrak{b}}(\mathfrak{c}, \mathsf{Op}\,\alpha(\vec{t}))) \in \mathcal{BS}\}$$
$$\Pi_{[\mathfrak{b},\mathfrak{c}]}^{\text{state}}(\mathcal{BS}) =_{def} \{\quad \chi \quad \mid \boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \boldsymbol{\mathcal{B}}_{\mathfrak{b}}(\mathfrak{c}, \chi)) \in \mathcal{BS}\ \}.$$

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

Example 33 (Belief Status Sets for RAMP-Agents) *Consider the map*

of heli1 *given in Example 29*

$$\begin{aligned}\mathcal{BS}(\texttt{heli1}) =_{def} \{\ & \mathbf{P}\mathit{fly}(\texttt{pointA},\texttt{pointB},10000,200), \mathbf{O}\mathit{fly}(\texttt{pointA},\texttt{pointB},10000,200),\\ & \boldsymbol{\mathcal{B}}_{\texttt{heli1}}(\texttt{heli2}, \mathbf{P}\mathit{fly}(\texttt{PointA},\texttt{PointB},10000,200)),\\ & \boldsymbol{\mathcal{B}}_{\texttt{heli1}}(\texttt{heli2}, \mathbf{in}(\texttt{pos}, \texttt{heli2} : \mathit{getPos}())),\\ & \boldsymbol{\mathcal{B}}_{\texttt{heli1}}(\texttt{heli2}, \boldsymbol{\mathcal{B}}_{\texttt{heli2}}(\texttt{tank1}, \mathbf{in}(\texttt{pos}, \texttt{tank1} : \mathit{getPos}()))))\\ & \boldsymbol{\mathcal{B}}_{\texttt{heli1}}(\texttt{heli2}, \boldsymbol{\mathcal{B}}_{\texttt{heli2}}(\texttt{tank1}, \mathbf{P}\mathit{drive}(\texttt{pointX},\texttt{pointY},40)))\}\end{aligned}$$

This belief status set for heli1 *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:

$$\begin{aligned}&\Pi^{\text{action}}_{\texttt{heli2}}(\mathcal{BS}(\texttt{heli1})) = \{\mathbf{P}\mathit{fly}(\texttt{pointA},\texttt{pointB},10000,200)\}\\ &\Pi^{\text{state}}_{\texttt{heli2}}(\mathcal{BS}(\texttt{heli1})) = \{\mathbf{in}(\texttt{pos}, \texttt{heli2} : \mathit{getPos}())\}\\ &\Pi^{\text{action}}_{[\texttt{heli2},\texttt{tank1}]}(\mathcal{BS}(\texttt{heli1})) = \{\mathbf{P}\mathit{drive}(\texttt{pointX},\texttt{pointY},40)\}\\ &\Pi^{\text{state}}_{[\texttt{heli2},\texttt{tank1}]}(\mathcal{BS}(\texttt{heli1})) = \{\mathbf{in}(\texttt{pos}, \texttt{tank1} : \mathit{getPos}())\}\end{aligned}$$

These sets formalize the following:

- $\Pi^{\text{action}}_{\texttt{heli2}}(\mathcal{BS}(\texttt{heli1}))$ *describes* heli1*'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.*
- $\Pi^{\text{state}}_{\texttt{heli2}}(\mathcal{BS}(\texttt{heli1}))$ *describes* heli1*'s beliefs about* heli2*'s state and it says that its position is* pos*.*
- $\Pi^{\text{action}}_{[\texttt{heli2},\texttt{tank1}]}(\mathcal{BS}(\texttt{heli1}))$ *describes* heli1*'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.*
- $\Pi^{\text{state}}_{[\texttt{heli2},\texttt{tank1}]}(\mathcal{BS}(\texttt{heli1}))$ *describes* heli1*'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 also needs to guess b's

action base, action program as well as the action and integrity constraints used by b. This is like guessing b's software package which we motivated and illustrated just before Definition 16 on page 18 (see the notion of *compatible* code call condition). We can easily accomplish this by adding an extra column to the belief table of each agent. This new column is called "Item" and all values in this column are in the following set of strings: { state, ic, ac, prog}. If, for instance, a's belief table contains a quadruple of the form $(\mathsf{b}, \phi, \chi, ac)$, then this may be read as saying that agent a believes that ϕ is true in agent b's action constraint if the condition χ holds. Similar readings are possible when the last column has the entry state, ic, etc. For notational convenience and better readability we merge all these ingredients into a set $\Gamma^{\mathsf{a}}(\mathsf{b})$.

Definition 34 ($\Gamma^{\mathsf{a}}(\mathsf{b})$, $\mathsf{Info}(\mathsf{a})$) *For agents $\mathsf{a}, \mathsf{b} \in \mathsf{A}$, 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* agent 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 $\sigma = [\mathsf{b}, \sigma_1]$ (I.e. a sequence σ whose head is b and whose tail is σ_1 instead of b: we therefore also use the notation $\Gamma^{\mathsf{a}}([\mathsf{b}, \sigma_1])$. $\Gamma^{\mathsf{a}}(\sigma)$ represents a's beliefs about b's beliefs about σ_1 (which itself may represent the beliefs of some agent c about yet another agent, and so on).*

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 agent program $\mathcal{P}$, the integrity constraints $\mathcal{IC}$ and action constraints $\mathcal{AC}$ used by agent a. Thus we define $\mathsf{Info}(\mathsf{a}) =_{def} \Gamma^{[]}(\mathsf{a})$.

Note that although not all of these functions will be referred to explicitly in the rest of this paper, they are important. Often we do not explicitly mention the set of underlying action constraints, integrity constraints, action base or the software package that agent a believes agent b to have (or that agent a believes w.r.t. a sequence σ), just for simplifying the notation.

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

Definition 35 (Extended Code Calls, $\mathcal{S}^{\mathbf{ext}}$) *Henceforth, given an agent a, we will 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) a : belief_table(), which returns the full belief table of agent a, as a set of triples $\langle \mathsf{h}, \phi, \chi_{\mathcal{B}} \rangle$,
(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* $\langle \mathfrak{h}, \mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{h}} \rangle$,

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

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

(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 functions which simulate 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.* "$\mathit{Op}\,\alpha$" *for action status atoms) in order to distinguish them from those that have to be checked in* $\mathfrak{a}$*'s state.*

(j) $\mathfrak{a}$: *not_bel_ccc_act*(σ), *which returns all the code call conditions and action status atoms that* $\mathfrak{a}$ *does not believe to be true in* σ*'s state.*

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

The functions introduced in the previous definition will be used in the definition of the mapping $\mathfrak{Trans}$ and the formulation of the properties corresponding to belief table and belief semantics compatibility (see Diagram 7 on page 54). They also play a role in the formulation of our main Theorem 58 on page 56 (although we often avoid explicitly mentioning integrity constraints, action constraints, the action base, and the whole software package when we consider an agent's state).

5.1 Feasible Belief Status Sets

Consider now an agent $\mathfrak{a}$ with associated structures, $\mathsf{Info}(\mathfrak{a})$. Suppose $\mathcal{BS}$ is an arbitrary status set. We would first like to identify the conditions that determine whether it "makes sense" for agent $\mathfrak{a}$ to hold the set of beliefs prescribed by $\mathcal{BS}$. In particular, agent $\mathfrak{a}$ must use some epistemically well justified criteria to hold a set, $\mathcal{BS}$, of beliefs. In this section, we introduce the concept of a *feasible belief status set*. Intuitively, $\mathcal{BS}$ 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:** $\mathcal{BS}$ must not contain any inconsistencies. For example, $\mathcal{BS}$ 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 Todo $= \{\alpha \mid \mathbf{Do}\,\alpha \in \mathcal{BS}\}$, then for each ground instance of an action constraint of the form ActSet $\hookleftarrow \chi$, either χ is false in the current agent state, or ActSet $\not\subseteq$ 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 coherence:** This condition requires that for any agent $\mathfrak{b}$, every induced status set $\Pi^{\text{action}}_{\mathfrak{b}}(\mathcal{BS})$ is feasible (in the original sense) with respect to the induced state $\Pi^{\text{state}}_{\mathfrak{b}}(\mathcal{BS})$ and agent 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 $\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{b}}$.
8. **Consistency of the Beliefs:** A (consistent) belief status set should only exist if there are no inconsistencies in the belief tables (an agent can erroneously hold the belief $\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{F}\alpha)$ as well as $\neg\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \mathbf{F}\alpha)$ at the same time. Or, even if there is no explicit such inconsistency, it might result from the facts in the belief table together with the closure under the semantics in the belief semantics table.

We are now ready to formalize the above 8 basic conditions through a sequence of definitions.

Definition 36 (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 $\mathbf{O}\alpha \in \Pi_\sigma^{\text{action}}(\mathcal{B}S)$, *then* $\mathbf{W}\alpha \notin \Pi_\sigma^{\text{action}}(\mathcal{B}S)$.
(2) If $\mathbf{P}\alpha \in \Pi_\sigma^{\text{action}}(\mathcal{B}S)$, *then* $\mathbf{F}\alpha \notin \Pi_\sigma^{\text{action}}(\mathcal{B}S)$.
(3) If $\mathbf{P}\alpha \in \Pi_\sigma^{\text{action}}(\mathcal{B}S)$, *then* $\Pi_\sigma^{\text{state}}(\mathcal{B}S) \models Pre(\alpha)$ *(i.e.* α *is executable in* $\Pi_\sigma^{\text{state}}(\mathcal{B}S)$*).*

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

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.

The following definition specifies what it means for a belief status set to be deontically closed.

Definition 37 (Deontically Closed Belief Status Set) *Suppose* $\mathcal{B}S$ *is a belief status set held by agent* $\mathfrak{a}$. $\mathcal{B}S$ *is said to be deontically closed if and only if the following two conditions hold:*

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

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

Intuitively, $\mathcal{B}S$ is deontically closed if for every agent sequence σ, if agent $\mathfrak{a}$ believes that action α is obligatory w.r.t. sequence σ, then $\mathfrak{a}$ must also believe that α is permitted w.r.t. σ. The following lemma establishes that given any belief status set, there exists a unique minimal (w.r.t. set inclusion) superset of it that is deontically closed.

Lemma 38 *Suppose $\mathcal{BS}$ is a belief status set held by agent* $\mathfrak{a}$. *Then there is a belief status set denoted* $\mathbf{D\text{-}Cl}(\mathcal{BS})$ *such that:*

(1) $\mathcal{BS} \subseteq \mathbf{D\text{-}Cl}(\mathcal{BS})$ *and*
(2) $\mathbf{D\text{-}Cl}(\mathcal{BS})$ *is deontically closed and*
(3) Every other deontically closed superset of $\mathcal{BS}$ *is a superset of* $\mathbf{D\text{-}Cl}(\mathcal{BS})$.

PROOF. First note that the set of all ground status atoms satisfies conditions (1) and (2) above. So all we have to show is 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, $\bigcup \mathcal{BS}_i$ 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.

The unique minimal status set that is a superset of $\mathcal{BS}$ and which is deontically closed is called the deontic closure of $\mathcal{BS}$.

Definition 39 (Deontic Closure D-Cl($\mathcal{BS}$)) *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 smallest superset of $\mathcal{BS}$ that is deontically closed.*

Just as in the case of deontically closed sets, a belief status set is said to be action closed *if, by definition,* whenever agent $\mathfrak{a}$ believes action α is obligatory for some agent sequence, then it must also believe that that agent sequence will do it. Furthermore, if it believes action α is done by some agent sequence, then it must also believe that that agent sequence is permitted to do it.

Definition 40 (Action Closed Belief Status Set) *Suppose $\mathcal{BS}$ is a belief status set held by agent* $\mathfrak{a}$. *$\mathcal{BS}$ is said to be* action closed *if, by definition, the following conditions hold:*

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

As in the case of deontically closed belief status sets, if $\mathcal{BS}$ is any belief status set, there is a unique minimal (w.r.t. set inclusion) superset of $\mathcal{BS}$ which is action closed. This is the gist of the following lema.

Lemma 41 *Suppose $\mathcal{BS}$ is a belief status set held by agent* $\mathfrak{a}$. *Then there is a unique belief status set, denoted* $\mathbf{A\text{-}Cl}(\mathcal{BS})$, *such that:*

(1) $\mathcal{BS} \subseteq \mathbf{A\text{-}Cl}(\mathcal{BS})$ *and*
(2) $\mathbf{A\text{-}Cl}(\mathcal{BS})$ *is action closed and*

(3) There is no strict subset of $\mathbf{A\text{-}Cl}(\mathcal{BS})$ *satisfying the preceding two conditions.*

PROOF. We have to show that such a minimal extension of $\mathcal{BS}$ exists and follow the proof of the preceding 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, $\bigcup \mathcal{BS}_i$ 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.

In view of the above theorem, we call $\mathbf{A\text{-}Cl}(\mathcal{BS})$ the *action closure* of $\mathcal{BS}$.

Definition 42 (Action Closure A-Cl($\mathcal{BS}$)) *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 smallest superset of $\mathcal{BS}$ that is action closed.*

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.

Definition 43 (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.

Example 44 ($\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS})$ for RAMP) *We continue with Example 29 on page 31 and consider the following belief status set for* heli1*:*

$$\mathcal{BS}_1(\mathsf{heli1}) =_{def} \{\ \boldsymbol{\mathcal{B}}_{\mathrm{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pointB}, \mathtt{tank1} : \mathit{getPos}())),$$
$$\mathbf{P}\mathit{fly}(\mathtt{pointA}, \mathtt{pointC}, 5000, 100), \mathbf{P}\mathit{attack}(\mathtt{pointC}, \mathtt{pointB})\}$$

Then $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS}_1(\mathsf{heli1})) = \{\mathbf{P}\mathit{attack}(\mathtt{pointA}, \mathtt{pointB})\}$.

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. In addition the atoms* $\mathbf{P}\mathit{fly}(\mathtt{pointA}, \mathtt{pointC}, 5000, 100)$ *and* $\mathbf{P}\mathit{attack}(\mathtt{pointC}, \mathtt{pointB})$ *were not preserved because there are no rules to support them.*

Definition 45 (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,* $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS}) = \{\mathsf{Op}\,\alpha \mid \mathsf{Op}\,\alpha \in \mathcal{BS} \text{ where } \mathsf{Op} \in \{\mathbf{O}, \mathbf{W}, \mathbf{P}, \mathbf{F}, \mathbf{Do}\}\}$.

Intuitively, this definition says that when we restrict $\mathcal{BS}$ to the action status atoms associated with agent $\mathtt{a}$, 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 preceding 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

$$\{\mathsf{Op}\,\alpha \mid \mathsf{Op}\,\alpha \in \mathcal{BS}\} = \{\mathbf{P}\mathit{fly}(\mathtt{pointA}, \mathtt{pointC}, 5000, 100), \mathbf{P}\mathit{attack}(\mathtt{pointC}, \mathtt{pointB})\}.$$

However, supose we add the following two rules to heli1's program:

$$\mathbf{P}\ \mathit{fly}(\mathtt{pointA},\ \mathtt{pointC},\ 5000, 100) \leftarrow$$
$$\mathbf{P}\ \mathit{attack}(\mathtt{pointC},\ \mathtt{pointB}) \leftarrow$$

In this case, the belief status set $\mathcal{BS}_2(\mathsf{heli1})$ satisfies the program closure rule:

$$\mathcal{BS}_2(\mathsf{heli1}) =_{def} \{\ \boldsymbol{\mathcal{B}}_{\mathrm{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pointB}, \mathtt{tank} : \mathit{getPos}())),$$
$$\mathbf{P}\mathit{fly}(\mathtt{pointA}, \mathtt{pointC}, 5000, 100),$$
$$\mathbf{P}\mathit{attack}(\mathtt{pointC}, \mathtt{pointB}),$$
$$\mathbf{P}\mathit{attack}(\mathtt{pointA}, \mathtt{pointB})\ \}.$$

Then $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS}_2(\mathsf{heli1}))$ is given by the three action status atoms

$$\mathbf{P}attack(\mathtt{pointA}, \mathtt{pointB}),$$

$$\mathbf{P}attack(\mathtt{pointC}, \mathtt{pointB}),$$

$$\mathbf{P}fly(\mathtt{pointA}, \mathtt{pointC}, 5000, 100).$$

Finally, we have the state consistency:

Definition 46 (State Consistency) *We require that $\mathcal{O}'_{\mathcal{S}} \models \mathcal{IC}$, where $\mathcal{O}'_{\mathcal{S}} = apply(\mathbf{Do}\,(\mathcal{B}S), \mathcal{O}_{\mathcal{S}})$ is the state which results after concuerrently executing all actions in $\mathbf{Do}\,(\mathcal{B}S)$ on the state $\mathcal{O}_{\mathcal{S}}$.*

At this point, we have finished 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.

Definition 47 (Local Coherence) *A belief status set, $\mathcal{B}S$, 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 $\Pi^{\text{action}}_{\sigma}(\mathcal{B}S)$ is feasible in the sense of Definition 64 and (21; 19) with respect to the induced state $\Pi^{\text{state}}_{\sigma}(\mathcal{B}S)$ and agent program $\mathcal{P}^{\mathfrak{a}}(\sigma)$.*

$\mathcal{B}S$ is said to be locally coherent *if, by definition, $\mathcal{B}S$ 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 (21; 19). It is important to note that $\Pi^{\text{action}}_{\sigma}(\mathcal{B}S)$ is a set of action status atoms, and that $\Pi^{\text{state}}_{\sigma}(\mathcal{B}S)$ involves no belief literals, and $\mathcal{P}^{\mathfrak{a}}(\sigma)$ is an *agent program* with no belief modalities, as defined earlier on in Definition 15 on page 16. Here are a few examples of what it means for a belief status set held by agent $\mathfrak{a}$ to be locally coherent.

Example 48 (Local Coherence) *Let*

$$\begin{aligned}\mathcal{B}S(\mathtt{heli2}) =_{def} \{\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli2}}(\mathtt{heli1}, \mathbf{P}fly(\mathtt{PointA}, \mathtt{PointB}, 1000, 100)),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli2}}(\mathtt{heli1}, \mathbf{in}(100, \mathtt{heli1} : getSpeed())),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli2}}(\mathtt{heli1}, \mathbf{in}(1000, \mathtt{heli1} : getAltitude()))\ \}\end{aligned}$$

and let heli1*'s program as believed by* heli2 *(we denote it by $\mathcal{P}^{\mathtt{heli2}}(\mathtt{heli1})$) be:*

$\mathbf{P}\mathit{fly}(\mathtt{X},\mathtt{Y},\mathtt{A},\mathtt{S})) \leftarrow \mathbf{in}(\mathtt{S},\mathtt{heli1}:\mathit{getSpeed}()),\ \mathbf{in}(\mathtt{A},\mathtt{heli1}:\mathit{getAltitude}()).$

The set $\mathcal{BS}(\mathtt{heli2})$ *is locally coherent w.r.t. the sequence* $(\mathtt{heli2})$. *Notice that:*

$$\Pi^{\text{action}}_{\mathtt{heli1}}(\mathcal{BS}(\mathtt{heli2})) =_{def} \{\mathbf{P}\mathit{fly}(\mathtt{PointA},\mathtt{PointB},1000,100)\}$$

is feasible with respect to:

$\Pi^{\text{state}}_{\mathtt{heli1}}(\mathcal{BS}(\mathtt{heli2})) = \{\mathbf{in}(100,\mathtt{heli1}:\mathit{getSpeed}()),\mathbf{in}(1000,\mathtt{heli1}:\mathit{getAltitude}())\}.$

Let

$$\begin{aligned}\mathcal{BS}(\mathtt{tank1}) =_{def} \{\ & \boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli2},\boldsymbol{\mathcal{B}}_{\mathtt{heli2}}(\mathtt{heli1},\mathbf{P}\mathit{fly}(\mathtt{PointA},\mathtt{PointB},1000,100))),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli2},\boldsymbol{\mathcal{B}}_{\mathtt{heli2}}(\mathtt{heli1},\mathbf{in}(100,\mathtt{heli1}:\mathit{getSpeed}()))),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli2},\boldsymbol{\mathcal{B}}_{\mathtt{heli2}}(\mathtt{heli1},\mathbf{in}(1000,\mathtt{heli1}:\mathit{getAltitude}())))\\ \}\end{aligned}$$

and let the $\mathcal{P}^{\mathtt{tank1}}([\mathtt{heli2},\mathtt{heli1}])$ *be the program* tank1 *believes* heli2 *believes* heli1 *has:*

$\mathbf{P}\ \mathit{fly}(\mathtt{X},\mathtt{Y},\mathtt{A},\mathtt{S}) \leftarrow \mathbf{in}(\mathtt{S},\mathtt{heli1}:\mathit{getSpeed}()),\ \mathbf{in}(\mathtt{A},\mathtt{heli1}:\mathit{getAltitude}()).$

Then $\mathcal{BS}(\mathtt{tank1})$ *is locally coherent w.r.t. the sequence* $[\mathtt{heli2},\mathtt{heli1}]$. *Just like in the previous example:* $\Pi^{\text{action}}_{[\mathtt{heli2},\mathtt{heli1}]}(\mathcal{BS}(\mathtt{tank1})) = \{\mathbf{P}\mathit{fly}(\mathtt{PointA},\mathtt{PointB},1000,100)\}$ *is feasible with respect to:*

$\Pi^{\text{state}}_{[\mathtt{heli2},\mathtt{heli1}]}(\mathcal{BS}(\mathtt{tank1})) = \{\mathbf{in}(100,\mathtt{heli1}:\mathit{getSpeed}()),\mathbf{in}(1000,\mathtt{heli1}:\mathit{getAltitude}())\}.$

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 $\mathtt{a}$. This may be encoded through the following condition.

Definition 49 (Compatibility with $\mathbf{BT}^{\mathtt{a}}$) *Suppose* $\langle \mathtt{h},\phi,\chi_{\mathcal{B}}\rangle$ *is in* $\mathbf{BT}^{\mathtt{a}}$. $\mathcal{BS}$ *is said to be* compatible *with* $\langle \mathtt{h},\phi,\chi_{\mathcal{B}}\rangle$ *if, by definition, either*

(1) $\mathtt{h}_\mathit{part}(\chi_{\mathcal{B}})$ *is false w.r.t.* $\Pi^{\text{state}}_{\mathtt{h}}(\mathcal{BS})$ *or* $\mathtt{a}_\mathit{part}(\chi_{\mathcal{B}})$ *is false w.r.t.* $\mathtt{a}$*'s state* $\mathcal{O}_{\mathcal{S}}$, *or*

(2) (a) if ϕ *is a code call condition or an action status atom then* $\boldsymbol{\mathcal{B}}_{\mathtt{a}}(\mathtt{h},\phi) \in \mathcal{BS}$,

(b) if ϕ *is a negated belief literal of the form* $\neg\boldsymbol{\mathcal{B}}_{\mathtt{h}}(\mathtt{c},\psi)$, *then* $\boldsymbol{\mathcal{B}}_{\mathtt{h}}(\mathtt{c},\psi) \notin \mathcal{BS}$,

(c) if ϕ *is an arbitrary conjunction of belief literals,* $\phi = \phi_1 \wedge \ldots \wedge \phi_n$ *then (a) and (b) apply to all* ϕ_i *separately.*

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

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

Example 50 (Compatibility) *We continue with Table 3 on page 30. We define*

$$\mathcal{BS}(\mathtt{tank1}) =_{def} \{\boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli1}, \mathbf{in}(\mathtt{pos1}, \mathtt{heli1} : \mathit{getPos}()))\}.$$

$\mathcal{BS}$ ($\mathtt{tank1}$*) is compatible with* $\mathbf{BT}^{\mathfrak{a}}$.

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

$$\mathcal{BS}(\mathtt{tank1}) = \{\mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : \mathit{getPos}())\}.$$

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

$$\boldsymbol{\mathcal{B}}_{\mathtt{tank1}}(\mathtt{heli1}, \mathbf{in}(\mathtt{pos1}, \mathtt{heli1} : \mathit{getPos}()))$$

according to the definition of compatibility.

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

Definition 51 (Compatibility with $\mathbf{BSemT}^{\mathfrak{a}}$) *Suppose* $\langle \mathfrak{b}, \mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{b}} \rangle$ *is an entry in* $\mathbf{BSemT}^{\mathfrak{a}}$*, and suppose $\mathcal{BS}$ is a belief status set. Let* $\mathcal{BS}[\mathfrak{b}] = \{\chi \mid (\neg)\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \chi) \in \mathcal{BS}\}$. *$\mathcal{BS}$ is said to be* compatible *with* $\langle \mathfrak{b}, \mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{b}} \rangle$ *if, by definition,*

$$\{\chi' \mid \mathcal{BS}[\mathfrak{b}] \models_{\mathcal{BS}em^{\mathfrak{a}}_{\mathfrak{b}}} \chi'\} \subseteq \mathcal{BS}.$$

$\mathcal{BS}$ is said to be compatible *with* $\mathbf{BSemT}^{\mathfrak{a}}$ *if, by definition, $\mathcal{BS}$ is compatible with every entry in* $\mathbf{BSemT}^{\mathfrak{a}}$.

The last property we need is the consistency of the belief table and the belief semantics table.

Definition 52 (Belief Consistency) *We require that a belief status set is consistent with respect to beliefs hold by agent* $\mathfrak{a}$ *about other agents, i.e. that it does not contain a belief atom φ and its negation $\neg\varphi$. Such a set is called belief-consistent wrt. agent* $\mathfrak{a}$.

We are now ready to define feasible belief status sets.

Definition 53 (Feasible Belief Status Set) *A belief status set, $\mathcal{BS}$ held by agent* $\mathtt{a}$*, 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 8 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 $\mathbf{BT}^{\mathrm{a}}$, compatible with $\mathbf{BSemT}^{\mathrm{a}}$ and belief-consistent).*

We illustrate the concept of a feasible belief status set in the example below.

Example 54 (Feasible Belief Status Sets) *Let the map of* heli1 *be the following;*

$$
\begin{aligned}
\textbf{\textit{r1:}}\ & \mathbf{P}\mathit{attack}(\mathtt{P1},\mathtt{P2}) \leftarrow \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1},\mathbf{in}(\mathtt{P2},\mathtt{tank1}:\mathit{getPos}())),\\
& \qquad\qquad\qquad\quad \mathbf{P}\mathit{fly}(\mathtt{P1},\mathtt{P3},\mathtt{A},\mathtt{S}),\\
& \qquad\qquad\qquad\quad \mathbf{P}\mathit{attack}(\mathtt{P3},\mathtt{P2}).\\
\textbf{\textit{r2:}}\ & \mathbf{O}\mathit{attack}(\mathtt{P1},\mathtt{P2}) \leftarrow \mathbf{P}\mathit{flyRoute}([\mathtt{P0},\mathtt{P1},\mathtt{P2},\mathtt{P3}],\mathtt{S}),\\
& \qquad\qquad\qquad\quad \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1},\mathbf{in}(\mathtt{P2},\mathtt{tank1}:\mathit{getPos}())).\\
\textbf{\textit{r3:}}\ & \mathbf{P}\mathit{attack}(\mathtt{pos3},\mathtt{pos1}) \leftarrow\\
\textbf{\textit{r4:}}\ & \mathbf{P}\mathit{fly}(\mathtt{pos0},\mathtt{pos3},\mathtt{5000},\mathtt{100}) \leftarrow
\end{aligned}
$$

Let the set of $\mathcal{IC}$ be given as;

$$
\begin{aligned}
&\mathbf{in}(\mathtt{S},\mathtt{heli}:\mathit{getSpeed}())\ \&\ S < \mathtt{MaxSpeed}\\
&\mathbf{in}(\mathtt{A},\mathtt{heli}:\mathit{getAltitude}())\ \&\ A < \mathtt{MaxAltitude}
\end{aligned}
$$

and suppose the set $\mathcal{AC}$ contains the following action constraints:

$$\{\mathit{attack}(\mathtt{P1},\mathtt{P2}),\mathit{attack}(\mathtt{P3},\mathtt{P4})\} \hookleftarrow P1 = P3\ \&\ P2 \neq P4$$

Let

$$
\begin{aligned}
\mathcal{BS}(\mathtt{heli1}) =_{def} \{\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1},\mathbf{in}(\mathtt{pos1},\mathtt{tank1}:\mathit{getPos}())),\\
& \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2},\mathbf{in}(\mathtt{pos2},\mathtt{heli2}:\mathit{getPos}())),\\
& \mathbf{P}\mathit{fly}(\mathtt{pos0},\mathtt{pos3},\mathtt{5000},\mathtt{100}),\\
& \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2},\mathbf{P}\mathit{fly}(\mathtt{pos0},\mathtt{pos1},\mathtt{10000},\mathtt{200})),\\
& \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2},\mathbf{in}(\mathtt{200},\mathtt{heli2}:\mathit{getSpeed}())),\\
& \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2},\mathbf{in}(\mathtt{10000},\mathtt{heli2}:\mathit{getAltitude}())),\\
& \mathbf{P}\mathit{attack}(\mathtt{pos3},\mathtt{pos1}),\\
& \mathbf{P}\mathit{attack}(\mathtt{pos0},\mathtt{pos1})\ \}.
\end{aligned}
$$

Then, we have

$$\Pi^{\text{action}}_{\text{heli2}}(\mathcal{BS}(\text{heli1})) = \{\mathbf{P}\mathit{fly}(\text{pos0}, \text{pos1}, 10000, 200)\}$$
$$\Pi^{\text{state}}_{\text{heli2}}(\mathcal{BS}(\text{heli1})) = \{\mathbf{in}(200, \text{heli2} : \mathit{getSpeed}()), \mathbf{in}(10000, \text{heli2} : \mathit{getAltitude}())\}$$
$$\Pi^{\text{state}}_{\text{tank1}}(\mathcal{BS}(\text{heli1})) = \{\mathbf{in}(\text{pos1}, \text{tank1} : \mathit{getPos}())\}$$

Let the belief table of heli1 *be Table 4 on page 30. Finally let* $\mathcal{P}^{\text{heli1}}(\text{heli2})$ *be:*

$$\mathbf{P}\ \mathit{fly}(\text{X}, \text{Y}, \text{A}, \text{S}) \leftarrow \quad \mathbf{in}(\text{S}, \text{heli2} : \mathit{getSpeed}()),\ \mathbf{in}(\text{A}, \text{heli2} : \mathit{getAltitude}()).$$

Then, the belief status set $\mathcal{BS}(\text{heli1})$ *is feasible with respect to the above* map, $\mathcal{AC}$ *and* $\mathcal{IC}$. *Note that* $\mathcal{BS}(\text{heli1})$ *satisfies all of the 8 conditions. It is deontically and action consistent, because* $\mathcal{BS}(\text{heli1})$ *does not contain any inconsistencies; although* $\mathbf{P}\mathit{fly}(\text{pos0}, \text{pos3}, 5000, 100)$ *is in* $\mathcal{BS}(\text{heli1})$, *the action status atom* $\mathbf{F}\mathit{fly}(\text{pos0}, \text{pos3}, 5000, 100)$ *is not.*

$\mathcal{BS}(\text{heli1})$ *is deontically and action closed beacuse* $\Pi^{\text{action}}_{\text{heli2}}(\mathcal{BS}(\text{heli1}))$ *does not contain any inconistency. It is closed under the* map. *Note that the set* $\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{BS}_2(\text{heli1}))$ *is given by*

$$\{\mathbf{P}\mathit{attack}(\text{pos0}, \text{pos1}), \mathbf{P}\mathit{attack}(\text{pos3}, \text{pos1}), \mathbf{P}\mathit{fly}(\text{pos0}, \text{pos3}, 5000, 100)\}.$$

$\mathcal{BS}(\text{heli1})$ *is locally coherent with respect to the sequence* $[\text{heli2}]$. *Note that*

$$\Pi^{\text{action}}_{\text{heli2}}(\mathcal{BS}(\text{heli1})) = \{\mathbf{P}\mathit{fly}(\text{pos0}, \text{pos1}, 10000, 200)\}$$

is feasible with respect to

$$\Pi^{\text{state}}_{\text{heli2}}(\mathcal{BS}(\text{heli1})) = \{\mathbf{in}(200, \text{heli2} : \mathit{getSpeed}()), \mathbf{in}(10000, \text{heli2} : \mathit{getAltitude}())\}$$

Moreover, $\mathcal{BS}(\text{heli1})$ *is also locally coherent with respect to the sequence* $[\text{tank1}]$ *as* $\Pi^{\text{action}}_{\text{tank1}}(\mathcal{BS}(\text{heli1})) = \emptyset$.

$\mathcal{BS}(\text{heli1})$ *is compatible with* $\mathbf{BT}^{\text{heli1}}$. *The following belief status set is also feasible:*

$$\begin{aligned}\mathcal{BS}_1(\mathtt{heli1}) =_{def} \{\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : \mathit{getPos}())),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{pos2}, \mathtt{heli2} : \mathit{getPos}())),\\ & \mathbf{P}\mathit{fly}(\mathtt{pos0}, \mathtt{pos3}, \mathtt{5000}, \mathtt{100}),\\ & \mathbf{P}\mathit{flyRoute}([\mathtt{point1}, \mathtt{point2}, \mathtt{point3}, \mathtt{point4}], \mathtt{S}),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{P}\mathit{fly}(\mathtt{pos0}, \mathtt{pos1}, \mathtt{10000}, \mathtt{200})),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{200}, \mathtt{heli2} : \mathit{getSpeed}())),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{10000}, \mathtt{heli2} : \mathit{getAltitude}())),\\ & \mathbf{P}\mathit{attack}(\mathtt{pos3}, \mathtt{pos1}),\\ & \mathbf{P}\mathit{attack}(\mathtt{pos0}, \mathtt{pos1})\ \}.\end{aligned}$$

Now consider the following belief status set for heli1*, with the same* map*, belief table,* $\mathcal{AC}$ *and* $\mathcal{IC}$*.*

$$\begin{aligned}\mathcal{BS}_2(\mathtt{heli1}) =_{def} \{\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : \mathit{getPos}())),\\ & \mathbf{P}\mathit{fly}(\mathtt{pos0}, \mathtt{pos3}, \mathtt{5000}, \mathtt{100}),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{P}\mathit{fly}(\mathtt{pos0}, \mathtt{pos1}, \mathtt{10000}, \mathtt{200})),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{200}, \mathtt{heli2} : \mathit{getSpeed}())),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{10000}, \mathtt{heli2} : \mathit{getAltitude}())),\\ & \mathbf{P}\mathit{attack}(\mathtt{pos3}, \mathtt{pos1}),\\ & \mathbf{P}\mathit{attack}(\mathtt{pos0}, \mathtt{pos1})\ \}.\end{aligned}$$

$\mathcal{BS}_2(\mathtt{heli1})$ *is not feasible because it is not compatible with* $\mathbf{BT}^{\mathtt{heli1}}$*. This is because* $\mathcal{BS}_2(\mathtt{heli1})$ *does not contain* $\boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{pos2}, \mathtt{heli1} : \mathit{getPos}()))$*.*

The following belief status set of heli1 *is also not feasible:*

$$\begin{aligned}\mathcal{BS}_3(\mathtt{heli1}) =_{def} \{\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : \mathit{getPos}())),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{pos2}, \mathtt{heli2} : \mathit{getPos}())),\\ & \mathbf{P}\mathit{fly}(\mathtt{pos0}, \mathtt{pos3}, \mathtt{5000}, \mathtt{100}),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{P}\mathit{fly}(\mathtt{pos0}, \mathtt{pos1}, \mathtt{10000}, \mathtt{200})),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{200}, \mathtt{heli2} : \mathit{getSpeed}())),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{10000}, \mathtt{heli2} : \mathit{getAltitude}()))\}.\end{aligned}$$

The reason is that it is not closed under the map $\mathcal{BP}$*'s rules, as*

$$\mathbf{App}_{\mathcal{BP},\mathcal{O}}(\mathcal{B}S_3(\mathtt{heli1})) = \{\mathbf{P}\mathit{attack}(\mathtt{pos0},\mathtt{pos1})\}.$$

Finally,the following belief status set is also not feasible:

$$\begin{aligned}\mathcal{B}S_4(\mathtt{heli1}) =_{def} \{\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{tank1}, \mathbf{in}(\mathtt{pos1}, \mathtt{tank1} : \mathit{getPos}())),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{pos2}, \mathtt{heli2} : \mathit{getPos}())),\\ & \mathbf{P}\mathit{fly}(\mathtt{pos0},\mathtt{pos3},\mathtt{5000},\mathtt{100}),\\ & \mathbf{F}\mathit{fly}(\mathtt{pos0},\mathtt{pos3},\mathtt{5000},\mathtt{100}),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{P}\mathit{fly}(\mathtt{pos0},\mathtt{pos1},\mathtt{10000},\mathtt{200})),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{200}, \mathtt{heli2} : \mathit{getSpeed}())),\\ & \boldsymbol{\mathcal{B}}_{\mathtt{heli1}}(\mathtt{heli2}, \mathbf{in}(\mathtt{10000}, \mathtt{heli2} : \mathit{getAltitude}())),\\ & \mathbf{P}\mathit{attack}(\mathtt{pos3},\mathtt{pos1}),\\ & \mathbf{P}\mathit{attack}(\mathtt{pos0},\mathtt{pos1})\ \}.\end{aligned}$$

It is not deontically consistent as both $\mathbf{P}\alpha$ *and* $\mathbf{F}\alpha$ *is in* $\mathcal{B}S_4(\mathtt{heli1})$*, where* $\alpha = \mathit{fly}(\mathtt{pos0},\mathtt{pos3},\mathtt{5000},\mathtt{100})$.

5.2 Rational Belief Status Sets

The notion of a rational belief status set is a useful strengthening of feasible belief 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 is a feasible belief status set. However it is not minimal as there is no reason to believe $\mathbf{Do}\alpha$. Rational sets, defined below, rule out such non-minimal status sets:

Definition 55 (Groundedness; Rational Belief Status Set) *A belief status set* $\mathcal{B}S$ *which is locally coherent, compatible with* $\mathbf{BT}^{\mathrm{a}}$*, and compatible with* $\mathbf{BSemT}^{\mathrm{a}}$ *is* grounded*, if there exists no belief status set* $\mathcal{B}S'$ *strictly contained in* $\mathcal{B}S$ *(*$\mathcal{B}S' \subset \mathcal{B}S$*) such that* $\mathcal{B}S'$ *satisfies the following 3 conditions of a feasible belief status set as given in Definition 53: deontically and action consistent, deontically and action closed, closed under the* map $\mathcal{BP}$*'s rules.*

A belief status set $\mathcal{B}S$ *is a* rational status set*, if* $\mathcal{B}S$ *is a feasible belief status*

set and $\mathcal{BS}$ is grounded.

If we compare the above definition with the original definition of a belief status set (Definition 53), the reader will note that only the state consistency requirement is not explicitly required while minimizing $\mathcal{BS}'$. In contrast, the local 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, state consistency is guaranteed, because we check groundedness only for *feasible* belief status sets.

5.3 Reasonable Belief Status Sets

As shown in (21; 19) for programs without beliefs, rational belief 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 belief status sets: $S_1 = \{\mathbf{Do}\,(\alpha), \mathbf{P}(\alpha)\}$, and $S_2 = \{\mathbf{Do}\,(\beta), \mathbf{P}(\beta)\}$. S_2 seems less intuitive because there is no rule in the program to justify deriving $\mathbf{Do}\,(\beta)$.

This leads, in analogy to (21; 19) to the following notion:

Definition 56 (Reasonable Belief 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 $\mathsf{red}^{\mathcal{BS}}(\mathcal{BP}, \mathcal{O}_\mathcal{S})$, 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 belief status set *for $\mathcal{BP}$ w.r.t. $\mathcal{O}_\mathcal{S}$, if it is a reasonable belief status set of the (positive) meta agent program $\mathsf{red}^{\mathcal{BS}}(\mathcal{BP}, \mathcal{O}_\mathcal{S})$ with respect to $\mathcal{O}_\mathcal{S}$.*

6 How to Implement Meta-Agent Programs?

Meta-Agent Programs significantly extend agent programs by allowing us to reason about beliefs. However the *IMPACT*-platform developed at the University of Maryland efficiently implements a *polynomial class* of agent programs. This brings up the question of whether we can take advantage of this work. In fact, as we 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 35.*

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 35 on page 37: those also involve the new datatypes (belief table and belief semantics table). 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 in the belief table and/or in the belief semantics table. In fact, this was why we explicitly formulated condition (7.) on compatibility with the belief semantics table. Furthermore if the third entry in the belief table, the belief condition, is present, then the first two entries of this triple specify a belief that must hold. Therefore we use the additional function

$$\mathsf{a} : \mathit{bel_ccc_act}(\sigma),$$

which was introduced in Definition 35 on page 37 and thus *implement* belief atoms with extended code calls:

$$\mathbf{in}(\text{"}\chi\text{"}, \mathsf{a} : \mathit{bel_ccc_act}(\mathsf{c}))$$

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 $\mathfrak{a} : bel_ccc_act(\sigma)$ 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 $\mathfrak{Trans}$ comes in. We map

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

to

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

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:

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

where Sem is either the *feasible*, *rational* or *reasonable* belief status set semantics.

Definition 57 ($\mathfrak{Trans}$) *Recall (cf. Definition 18 on page 19) the maximal belief language $\mathcal{BL}^{\mathfrak{a}}_{\infty}$ associated with an agent $\mathfrak{a}$. We define the mapping*

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

by induction on the structure of belief literals:

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) $\mathfrak{Trans}(\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \phi)) \mapsto \mathbf{in}(\text{"}\phi\text{"}, \mathfrak{a} : bel_ccc_act(\mathfrak{b}))$,
(2) $\mathfrak{Trans}(\neg\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \phi)) \mapsto \mathbf{in}(\text{"}\phi\text{"}, \mathfrak{a} : not_bel_ccc_act(\mathfrak{b}))$,

***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*

$$\mathbf{in}(\text{"}\chi\text{"}, \mathfrak{a} : bel_ccc_act([\mathfrak{b}, \rho])) \textit{ if } \mathfrak{Trans}(\phi) = \mathbf{in}(\text{"}\chi\text{"}, \mathfrak{a} : bel_ccc_act(\rho))$$

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

$$\mathbf{in}(\text{"}\chi\text{"}, \mathfrak{a} : not_bel_ccc_act([\mathfrak{b}, \rho])) \textit{ if } \mathfrak{Trans}(\phi) = \mathbf{in}(\text{"}\chi\text{"}, \mathfrak{a} : bel_ccc_act(\rho))$$

Linear Extension to $\mathcal{BL}^{\mathfrak{a}}_{\infty}$: *Thus far $\mathfrak{Trans}$ is only defined on belief literals and not on arbitrary belief formulae (which can be arbitrary conjunctions of belief literals (see Definition 18 on page 19)). However we can easily extend $\mathfrak{Trans}$:*

$$\mathfrak{Trans}(bel_lit_1 \wedge \ldots \wedge bel_lit_n) = \mathfrak{Trans}(bel_lit_1) \wedge \ldots \wedge \mathfrak{Trans}(bel_lit_n).$$

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 60 for agent programs without beliefs.

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

$$\mathfrak{Trans}^{state}(\mathcal{BS}, \mathsf{b}) \;=_{def} \{\chi \mid \boldsymbol{\mathcal{B}}_{\mathrm{a}}(\mathsf{b}, \chi) \in \mathcal{BS} \text{ and } \chi \text{ is a code call condition}\}$$
$$\mathfrak{Trans}^{action}(\mathcal{BS}, \mathsf{b}) =_{def} \{\mathsf{Op}\,\alpha(\vec{t}) \mid \boldsymbol{\mathcal{B}}_{\mathrm{a}}(\mathsf{b}, \mathsf{Op}\,\alpha(\vec{t})) \in \mathcal{BS},\ ^{2}\}.$$

As in Definition 32 on page 34, these definitions are easily extended to arbitrary sequences σ instead of just b.

The transformation $\mathfrak{Trans}$ maps all belief literals into extended code call conditions and will be used to map any set containing belief literals (like belief status sets or meta agent programs) into sets without belief literals, but containing extended code calls. In addition, $\mathfrak{Trans}$ can be easily applied to $\mathcal{BP}$:

$\mathfrak{Trans}(\mathcal{BP})$**:** this is the program $\mathcal{BP}$ where all ocurrences of belief literals bel_lit have been replaced by $\mathfrak{Trans}$(bel_lit).

Thus $\mathfrak{Trans}$ 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 compatibility with the belief table and the belief semantics table (see Definitions 49 on page 45 and 22 on page 23). 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 can not expect a strong equivalence result to hold. The following picture may help to clarify the problem:

$$\begin{array}{lccc} & \mathcal{BP} & \xrightarrow{\mathfrak{Trans}} & \mathcal{P} \\ \text{Compatible with Belief Semantics Belief Table} & \uparrow \mathsf{Sem}^{\mathrm{new}} & & ??? \uparrow \mathsf{Sem}^{\mathrm{old}} \\ & \mathcal{BS} & \xrightarrow{\mathfrak{Trans}} & S \end{array} \tag{7}$$

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 $\mathcal{P}$. However, 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

[2] where $\mathsf{Op} \in \{\mathbf{O}, \mathbf{W}, \mathbf{P}, \mathbf{F}, \mathbf{Do}\}$

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 $\mathcal{S}^{\text{ext}}$,
(2) to add a new condition (which can not be expressed as an integrity constraint) which ensures the compatibility with the belief table.

To handle 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 $\mathsf{b} \in \mathsf{A}$):

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

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

$$\begin{aligned}
&\mathbf{in}(\langle \mathsf{b}, \mathcal{B}Sem^{\mathsf{a}}_{\mathsf{b}} \rangle, \mathsf{a} : \mathit{belief_sem_table}()) \quad \& \\
&\mathbf{in}(\mathbf{true}, \mathsf{a} : \mathit{bel_semantics}(\mathsf{b}, \text{"}\chi\text{"}, \text{"}\chi'\text{"})) \ \& \\
&\mathfrak{Trans}(\chi) \\
&\Rightarrow \\
&\mathfrak{Trans}(\chi')
\end{aligned}$$

To handle 2. we require the following condition:

Closure: Let the state $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfy $\mathbf{in}(\langle \mathsf{b}, \phi, \chi_{\mathcal{B}} \rangle, \mathsf{a} : \mathit{belief_table}())$ as well as $\mathsf{a_part}(\chi_{\mathcal{B}})$ and let $\mathfrak{Trans}^{\text{state}}(\mathcal{B}S, \mathsf{b})$ satisfy $\mathsf{b_part}(\chi_{\mathcal{B}})$. 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 $\mathbf{in}(\phi, \mathsf{a} : \mathit{bel_ccc_act}(\mathsf{b}))$.
(2) If ϕ is of the form $\boldsymbol{\mathcal{B}}_{\mathsf{b}}(\mathsf{c}, \phi')$, where ϕ' is a code call condition or an action status atom, then $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies $\mathbf{in}(\text{"}\phi'\text{"}, \mathsf{a} : \mathit{bel_ccc_act}([\mathsf{b}, \mathsf{c}]))$.

(3) If ϕ is of the form $\neg\boldsymbol{\mathcal{B}}_{\mathfrak{b}}(\mathfrak{c}, \phi')$, where ϕ' is a code call condition or an action status atom, then $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies $\mathbf{in}(\text{"}\phi'\text{"}, \mathfrak{a} : \mathit{not_bel_ccc_act}([\mathfrak{b}, \mathfrak{c}]))$.

(4) More generally, if ϕ is a nested belief atom, then we can associate with this atom a sequence $[\rho]$ (as introduced in Definition 31 on page 34) and we require that $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies $\mathbf{in}(\text{"}\phi'\text{"}, \mathfrak{a} : \mathit{bel_ccc_act}([\rho]))$. Similarly, if ϕ is a *negated* nested belief literal, we require that $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies $\mathbf{in}(\text{"}\phi'\text{"}, \mathfrak{a} : \mathit{not_bel_ccc_act}([\rho]))$.

Thus we end up with the following picture:

$$\begin{array}{lccc} & \mathcal{BP} & \xrightarrow{\mathfrak{Trans}} & \mathcal{P} \\ \text{Compatible with Belief Semantics Belief Table} & \uparrow \mathsf{Sem}^{\text{new}} & & \mathcal{IC}^{\text{ext}} \text{ Closure } \uparrow \mathsf{Sem}^{\text{old}} \\ & \mathcal{BS} & \xrightarrow{\mathfrak{Trans}} & S \end{array} \tag{8}$$

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

Theorem 58 (Implementing Belief Programs by Agent Programs) *Let $\mathcal{BP}$ be a meta agent program, $\mathfrak{a} \in \mathsf{A}$, $\mathcal{O}_{\mathcal{S}^{ext}}$ 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 $\mathcal{BS}$ is a feasible belief status set of agent $\mathfrak{a}$ wrt. $\mathcal{BP}$, $\mathcal{O}_{\mathcal{S}^{ext}}$, $\mathcal{IC}$ and $\mathcal{AC}$, then

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

*Moreover, every feasible status set of $\mathfrak{Trans}(\mathcal{BP})$ for a state $\mathcal{O}_{\mathcal{S}^{ext}}$ and $\mathcal{IC}^{ext}$ where $\mathcal{O}_{\mathcal{S}^{ext}}$ 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 consistency*, *(3) deontic and action closure* and *(4) state consistency*. All these properties are immediate from the corresponding conditions for $\mathcal{BS}$ (see **S1** of Definition 64 for *(1)*, Definition 61 for *(2)*, Lemmas 61 and 62 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 com-*

patibility.

The condition 2. is implied by *local coherence.*

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 $\mathfrak{a} : bel_ccc_act([\rho])$. Whenever $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies a code call atom

$$\mathbf{in}(\text{"}\chi\text{"}, \mathfrak{a} : bel_ccc_act([\mathfrak{b}, \rho]))$$

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 $\mathcal{BS}^{\text{new}}$. Whenever $\mathcal{O}_{\mathcal{S}^{\text{ext}}}$ satisfies a code call atom

$$\mathbf{in}(\text{"}\chi\text{"}, \mathfrak{a} : not_bel_ccc_act([\mathfrak{b}, \rho]))$$

where "χ" is a code call atom of the form "$\mathbf{in}(\phi, :())$" or an action status atom, then we add $\neg\boldsymbol{\mathcal{B}}_{\mathfrak{a}}(\mathfrak{b}, \chi)$ 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 compatibility condition. By construction, $\mathcal{BS}^{\text{new}}$ is a status set and the feasibility is guarantueed by the feasiblity of S and the conditions we have just mentioned.

Corollary 59 (Invariance of $\mathfrak{Trans}(\mathcal{BP})$) *If $\mathcal{BS}$ is a rational (resp. reasonable) belief status set of agent $\mathfrak{a}$ wrt. $\mathcal{BP}$, $\mathcal{O}_{\mathcal{S}^{ext}}$, $\mathcal{IC}$ and $\mathcal{AC}$, then*

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

(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 $\mathbf{in}(\mathcal{P}^{\mathfrak{a}}(\sigma), \mathfrak{a} : action_program(\sigma))$ is true in $\mathcal{O}_{\mathcal{S}^{ext}}$.

Moreover, every rational *(resp.* reasonable*) status set of $\mathfrak{Trans}(\mathcal{BP})$ for a state $\mathcal{O}_{\mathcal{S}^{ext}}$ and $\mathcal{IC}^{ext}$ where $\mathcal{O}_{\mathcal{S}^{ext}}$ 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 58, it only remains to prove that every $\mathcal{BS}$ which is locally coherent, compatible with $\mathbf{BT}^{\mathfrak{a}}$ and with $\mathbf{BSemT}^{\mathfrak{a}}$ 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 43 for programs with beliefs and Definition 63 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 be 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. * □

Theorem 58 and Corollary 59 are *extremely* important. In effect, they say that dealing with beliefs can be reduced to dealing with the specific belief related data structures (belief table and belief semantics table) introduced in this paper. As a consequence, a system for meta-agent programs does not have to be implemented from scratch—instead, the *IMPACT* agent program development environment developed in (20) may be used directly as a computational engine for processing agents that reason with beliefs. (20) describes an implementation of ordinary agent programs (not maps) for a class of agent programs called *regular agent programs* which are guaranteed to be polynomially evaluable. Regular agent programs generalize the well known concept of stratification in logic programming. It is possible to define a notion of a regular map in such a way that a map $\mathcal{BP}$ is considered to be regular if and only if the regular agent program $\mathfrak{Trans}(\mathcal{BP})$ is regular. The effect of this would be that for regular maps, computation of feasible belief status sets can be done polynomially—the reader can easily verify that $\mathfrak{Trans}(\mathcal{BP})$ is polynomially computable.

7 Strengths and Shortcomings

An agent designer will use meta-agent programs for specifying the operating principles of his agent in those cases where agents need to reason with beliefs about other agents' states and/or actions. In this section, we overview the strengths and weaknesses of maps for this purpose.

Strengths:

- maps work with data stored in arbitrary data structures and can be built on top of existing legacy code or code developed in specialized languages. No theory of agenthood is likely to be of practical use without this property, as most real world data is not stored in logical form.
- An agent designer can use a map to couple arbitrary actions to the operating principles of agent—in other words, arbitrary actions can be coupled to agent state changes. Such actions can be mundane actions such as creating files, mailing files, executing queries, updating data, or supposedly "more intelligent" actions such as updating beliefs, resolving contradictions, creating plans and schedules, etc.
- One consequence of the above facts is that our framework is *highly flexible*—the semantics of maps should not and does not depend upon a specific belief update mechanism or a specific contradiction removal mechanism,but rather uses these as "pluggable" parameters. This is theoretically clean and practically useful because different applications are likely to use different belief update mechanism (or in many cases none at all) and different contradiction removal mechanisms. An agent system should allow the designer this flexibility, rather than imposing a predetermined choice on him (though such predetermined options may well reside in a library such as an action library we are building in *IMPACT*).
- Another major strength of our framework is the transformation $\mathfrak{Trans}$ which converts any map in polynomial time into an ordinary agent program. As described in (20) and (51, Chapter 13), there is a current existing implementation of a polynomial class of ordinary agent programs that has been shown to scale up well to large data sets (e.g. a logistics application built in *IMPACT* computes joins involving approximately 49 billion tuples over heterogeneous Oracle and multi-record nested file structures in about 40 seconds over a network).Note that this implementation is a first order implementation, not propositional (as is the case with almost all implementations of belief logics).
- It is important to note that the designer of an agent $\mathfrak{a}$ may choose to make the agent program, integrity constraints, action constraints etc. of agent part of $\mathfrak{a}$'s state if he so desires. Thus the agent's behavior can be autonomously modified by tha agent itself if the agent designer provides $\mathfrak{a}$ with actions to do so.

Weaknesses: The proposed map paradigm also has some weaknesses.

- The first weakness is that agent $\mathfrak{a}$ guesses an agent program of agent $\mathfrak{b}$ rather than guessing a meta-agent program of $\mathfrak{b}$. Such an extension is challenging becuase it makes the semantics of maps computationally much more complex and in all likelihood, will result in the use of Kripke-style model theory. The computational complexity of such semantics are very high even in the propositional, and first order belief logics are highly undecidable (see (22)). Thus, there is a tradeoff between expressiveness

and implementability, which we made.

- It may also be argued that agents may hold disjunctive beliefs or probabilistic beliefs about other agents. This cannot be modeled in our current framework. Again, it is well known that disjunctive reasoning is computationally more expensive than definite reasoning—hence, future research may focus on including disjunctive or probabilistic beliefs in maps *without* increasing the computational complexity of map computations.
- We have not given explicit mechanisms for belief updating and contradiction removal in this paper. Though this may be considered a weaknesses, one strength of our framework is its ability to "plug in" such mechanisms (several of which have been developed by researchers across the world) into maps directly. maps provide a framework that builds on top of such modules that may be selected by the agent designer, rather than ramming them down an agent designer's unwilling throat.

8 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 (21; 19) with other agent systems, as that has been done in great detail in (21). Rather, we focus primarily on meta-reasoning capabilities of agents and compare maps with meta reasoning capabilities of other agent frameworks.

In (32) The authors 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

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

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 (32), 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 (31).

(46) showed 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 (2),

and inference machine that embodies the REVISE algorithm (17) for eliminating contraditions, and a control layer. No meta-reasoning issues are brought up explicitly in this work.

Concurrently with our effort, the authors of (38; 37) 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.

(41) was one of the first proposals of a formal extension of auto-epistemic logic to deal with multiagent reasoning. She extended auto-epistemic logic (40) 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") (17). 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 (42).

(25) describes 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 (26), 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). (23) presents 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.

In contrast to the above bodies of work, we believe that our agent framework provides a unifying theoretical framework for a variety of AI research including many of those listed above. Our architecture:

- *Builds on arbitrary data structures*—a necessary criterion for any practical application where lots of legacy code already exists and/or where data structures that are best suited for the application must be used to support the application.
- *Allows agents to use whatever actions they need*—for example, one agent may use one conflict resolution policy and/or belief updating policy, while another agent may use a different one. Resolving conflicts, updating beliefs, performing updates, inductive learning, planning, etc.—all topics well studied in logic programming—are all usable within *IMPACT* agents as actions!
- *Allows agents to create new actions from existing ones*—we are all aware that agents often do many things at the same time. An agent that is traveling from one location to another can update its beliefs at the same time. The notion of concurrency introduced in this paper is very generic, and allows agents to mix and match actions to produce new ones!
- *Allows agents to reason as little or as much about beliefs as they need*—different applications will require different levels of sophistication in the reasoning used by agents. For instance, the coordination agent in the RAMP example needs to perform sophisticated reasoning about its beliefs about other agents and the actions they may or may not perform. On the other hand, a relatively "dumb' database agent may only need to know how to respond to user queries. Such an agent may not need sophisticated belief capabilities. Our framework allows both.

A variety of open problems exist. These include the ability to perform actions over widely varying temporal intervals (see (51, Chapter 8)), performing actions in the presence of different types of uncertainty, and scheduling complex actions. Though time, uncertainty, etc. can all be "pushed" into the agent's state, techniques are needed to handle them inside such states. For example, the study of uncertainty in relational databases is far from mature (cf. (34; 16)). Extending such models of uncertainty to arbitrary data structures is a major challenge. Likewise, while temporal reasoning in logic is well studied, temporal reasoning over arbitrary data structures is not well understood (though temporal reasoning in relational databases is well understood (53, Chapters 4-6)). Thus, extending temporal data models to diverse data structures is an important priority. The mix of time and uncertainty is even more complex—to our knowledge the only published paper on this topic (18) represents an excellent start on probabilistic temporal databases, but has many restrictive assumptions, some of which are removed in (15). Extending probabilistic temporal reasoning to diverse data structures poses yet another challenge, especially when actions that make decisions based on such data must be coupled to the data structures (see (51, Chapter 9)).

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 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 `http://www.cs.umd.edu/~vs/agent/impact.html`.

As beliefs that an agent a may hold about another agent 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 a can estimate what agent b may do in the future, and to reason about how agent b's beliefs may evolve in the future.

Acknowledgments

We gladly appreciate the careful reading of the four referees of this paper. They provided us with lots of helpful remarks and pointed out numerous inconsistencies of the first and second draft of this paper. We also thank Fatma Özcan for an extensive proofreading and for putting together the appendix. Needless to say that any remaining inconsistencies are at our responsibility.

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

[1] S. Adali and R. Emery. A Uniform Framework for Integrating Knowledge in Heterogeneous Knowledge Systems. In P. S. Yu and A. L. P. Chen, editors, *Proceedings of the Eleventh International Conference on Data Engineering*, pages 513–520, Taipei, Taiwan, March 1995. IEEE Computer Society.

[2] J. J. Alferes and L. M. Pereira. Reasoning with Logic Programming. In *Springer Lecture Notes in Artificial Intelligence*, volume 1111, 1996.

[3] L. Åquist. Deontic Logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Vol.II*, chapter II.11, pages 605–714. D. Reidel Publishing Company, 1984.

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

[5] K. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus. IMPACT: A Platform for Collaborating Agents. *IEEE Intelligent Systems*, 14:64–72, March/April 1999.

[6] R. Bayardo et al. Infosleuth: Agent-based Semantic Integration of Information in Open and Dynamic Environments. In J. Peckham, editor, *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 195–206, Tucson, Arizona, 1997.

[7] John Benton and V. S. Subrahmanian. Using Hybrid Knowledge Bases for Missile Siting Problems. In IEEE Computer Society, editor, *Proceedings of the Conference on Artificial Intelligence Applications*, pages 141–148, 1994.

[8] P. Bonatti, S. Kraus, J. Salinas, and V. S. Subrahmanian. Data Security in Heterogenous Agent Systems. In M. Klusch, editor, *Cooperative Information Agents*, pages 290–305. Springer, 1998.

[9] Stefan Brass, Jürgen Dix, and Teodor Przymusinski. Computation of the semantics of autoepistemic belief theories. *Artificial Intelligence*, 112(1-2), 1999.

[10] Stefan Brass, Jürgen Dix, Ilkka Niemelae and Teodor Przymusinski. On the Equivalence of the Static and Disjunctive Wellfounded Semantics and its Computation. *Theoretical Computer Science*, to appear.

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

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

[13] R. G. G. Cattell et al. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1997.

[14] S. Chawathe et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *IPSJ Conference*, Tokyo, Japan, October 1994. Also available via anonymous FTP from host db.stanford.edu, file /pub/chawathe/1994/tsimmis-overview.ps.

[15] A. Dekhtyar, R. Ross, and V. S. Subrahmanian. Probabilistic Temporal Databases, I: Algebra. To appear in *Journal of Logic Programming*, 1999.

[16] D. Dey and S. Sarkar. A Probabilistic Relational Model and Algebra. *ACM Transactions on Database Systems*, 21(3):339–369, 1996.

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

[18] C. Dyreson and R. Snodgrass. Supporting Valid-Time Indeterminacy. *ACM Transactions on Database Systems*, 23(1):1–57, 1998.

[19] T. Eiter and V. S. Subrahmanian. Heterogeneous Active Agents, II: Algorithms and Complexity. *Artificial Intelligence*, 108(1-2):257–307, 1999.

[20] T. Eiter, V. S. Subrahmanian, and T. J. Rogers. Heterogeneous Active Agents, III: Polynomially Implementable Agents. submitted for journal publication, 1999.

[21] Thomas Eiter, V. S. Subrahmanian, and Georg Pick. Heterogeneous Active Agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, 1999.

[22] R. Fagin and J. Halpern. Uncertainty, Belief and Probability. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, IJCAI 89*, pages 1161–1167, Detroit, MI, USA, August 1989. Morgan Kaufmann.

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

[24] Peter W. Foltz and Susan T. Dumais. Personalized information delivery: An analysis of filtering methods. In *Proceedings of ACM CHI'92 Con-*

ference on Human Factors in Computing Systems – Posters and Short Talks, Posters: Designing for Use, 1992.

[25] P. Gmytrasiewicz and E. Durfee. A Logic of Knowledge and Belief for Recursive Modeling. In *Proceedings of AAAI-92*, pages 628–634. Morgan Kaufman, 1992.

[26] P. Gmytrasiewicz, E. Durfee, and D. Wehe. A Decision-Theoretic Approach to Coordinating Multiagent Interactions. In *Proceedings of IJCAI 1991*, pages 62–68. Morgan Kaufman, 1991.

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

[28] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. J. Ch. Meyer. Formal Semantics of an Abstract Agent Programming Language. In *Intl. Workshop on Agent Theories, Architectures, and Languages*, pages 204–218, Providence, RI, 1997.

[29] S. Ishizaki. Multiagent Model of Dynamic Design: Visualization as an Emergent Behavior of Active Design Agents. In M. Huhns and M. Singh, editors, *Readings in Agents*, pages 172–179. Morgan Kaufmann Press, 1997.

[30] S. Kanger. Law and Logic. *Theoria*, 38, 1972.

[31] R. Kowalski. Using metalogic to reconcile reactive with rational agents. In *Meta-Logics and Logic Programming*. MIT Press, 1995.

[32] R. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence, Special Issue on* Logic Programming and Nonmonotonic Reasoning*, edited by J. Dix and J. Lobo*, 25(3–4), 1999.

[33] Y. Labrou and T. Finin. Semantics for an Agent Communication Language. In *Intl. Workshop on Agent Theories, Architectures, and Languages*, pages 199–203, Providence, RI, 1997.

[34] V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. ProbView: A Flexible Probabilistic Database System. *ACM Transactions on Database Systems*, 22(3):419–469, September 1997.

[35] J.W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1984, 1987.

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

[37] M. Martelli, V. Mascardi, and F. Zini. CaseLP: a Complex Application Specification Environment based on Logic Programming. In *Proc. of ICLP'97 Post Conference Workshop on Logic Programming and Multi-Agents*, pages 35–50, Leuven, Belgium, 1997.

[38] M. Martelli, V. Mascardi, and F. Zini. 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)*, pages 331–354, London, UK, 1998.

[39] J.-J. Ch. Meyer and R.J. Wieringa (eds.). *Deontic Logic in Computer Science*. Wiley & Sons, Chichester et al, 1993.

[40] R.C. Moore. Semantical Considerations on Nonmonotonic Logics. *Artificial Intelligence*, 25:75–94, 1985.

[41] L. Morgenstern. A Formal Theory of Multiple Agent Nonmonotonic Reasoning. In *Proceedings of AAAI-90*, pages 538–544, 1990.

[42] A. S. Rao and M. Georgeff. Modeling Rational Agents within a BDI-Architecture. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the International Conference on Knowledge Representation and Reasoning*, pages 473–484, Cambridge, MA, 1991. Morgan Kaufmann.

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

[44] S. J. Rosenschein and L. P. Kaelbling. A Situated View of Representation and Control. *Artificial Intelligence*, 73:149–173, 1995.

[45] D. Rus, R. Gray, and D. Kotz. Transportable Information Agents. In M. Huhns and M. Singh, editors, *Readings in Agents*, pages 283–291. Morgan Kaufmann Press, 1997.

[46] M. Schroeder, I. de Almeida Mora, and L. M. Pereira. A Deliberative and Reactive Diagnosis Agent based on Logic Programming. In M.J. Wooldridge J.P. Muller and N. Jennings, editors, *Intelligent Agents III: Lecture Notes in Artificial Intelligence Vol. 1193*, pages 293–307. Springer Verlag, 1997.

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

[48] Y. Shoham. Agent Oriented Programming. *Artificial Intelligence*, 60:51–92, 1993.

[49] J. Siegal. *CORBA Fundementals and Programming*. John Wiley and Sons, New York, 1996.

[50] Jean-David Sta. 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, pages 177–178, 1993.

[51] V.S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Özcan, and Robert Ross. *Heterogenous Active Agents*. MIT-Press, 1999. ISBN 0262194244.

[52] F. Wilder. *A Guide to the TCP/IP Protocol Suite*. Artech House, 1993.

[53] C. Zaniolo, S. Ceri, C. Faloutsos, R. T. Snodgrass, V. S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann, 1997.

A Agent Programs without Beliefs

The following definitions are taken from (21).

A.1 Feasible, Rational and Reasonable Semantics

Definition 60 (Status Set) *A* status set *is any set S of ground action status atoms over $\mathcal{S}$. For any operator $\mathsf{Op} \in \{\mathbf{P}, \mathbf{Do}, \mathbf{F}, \mathbf{O}, \mathbf{W}\}$, we denote by $\mathsf{Op}(S)$ the set $\mathsf{Op}(S) = \{\alpha \mid \mathsf{Op}(\alpha) \in S\}$.*

Definition 61 (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 $\mathcal{O}_{\mathcal{S}} \models \exists^* Pre(\alpha)$, where $\exists^* Pre(\alpha)$ denotes the existential closure of $Pre(\alpha)$, i.e., all free variables in $Pre(\alpha)$ 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.

Definition 62 (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.

Definition 63 (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, $\boldsymbol{App}_{\mathcal{P},\mathcal{O}_\mathcal{S}}(S)$ 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 $r : A \leftarrow L_1, \ldots, L_n$ 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 $\mathsf{Op}(\alpha) \in B^+(r) \cup \{A\}$ such that $\mathsf{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$, the action α is executable in state $\mathcal{O}_\mathcal{S}$.

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.

Definition 64 (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 $\mathcal{O}_\mathcal{S}$.*

Definition 65 (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.*

Definition 66 (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 tank Agents

B.1.1 Code Calls

(1) Drive forward at speed `Speed` (`0` to `Max` speed)
`tank`: *goForward*(`Speed`) → `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`: *getPos*() → `2DPoint`

(6) Get current heading
`tank`: *getHeading*() → `Heading`

(7) Aim the gun at 3D point `Point`
`tank`: *aim*(`Point`) → `Boolean`

(8) Fire the gun using the current aim
`tank`: *fire*() → `Boolean`

(9) Compute the distance between two 2D points
`tank`: *computeDistance*(`X`, `Y`) → `Distance`

(10) Retrieve the maximum range for the gun
`tank`: *getMaxGunRange*() → `Distance`

(11) Calculate the next position of the tank when driving with `Speed` from `CurrentLocation` `tank`: *calculateNextPos*(`CurrentLocation`, `Speed`) → `2DPoint`

B.1.2 Actions

(1) Drive from to 2D point `From` to 2D point `To` at speed `Speed`
Name: *drive*(`From`, `To`, `Speed`)
Pre: **in**(`From`, `tank` : *getPos*())
Del: **in**(`From`, `tank` : *getPos*())
Add: **in**(`CurrentLocation`, `tank` : *getPos*())&

in(`To`, `tank` : *calculateNextPos*(`CurrentLocation`, `Speed`))

(2) Drive route `Route` given as a sequence of 2D points at speed `Speed`
Name: *driveRoute*(`Route`, `Speed`)
Pre: **in**(`Route(0).Pos`, `tank` : *getPos*())
Del: **in**(`Route(0).Pos`, `tank` : *getPos*())
Add:

in(`Route(Route.Count).Pos`, code call)

where code call is

`tank` : *calculateNextPos*(`Route(Route.Count` − `1).Pos`, `Speed`).

(3) Attack vehicle at position `Pos` from position `MyPos`
Name: *attack*(`MyPos`, `Pos`)
Pre: **in**(`MyPos`, `tank` : *getPos*())&

in(`Distance`, `tank` : *computeDistance*(`MyPos`, `Pos`))&

in(`maxRange`, `tank` : *getMaxGunRange*())&`Distance` < `maxRange`
Del: {}
Add: {}

B.2 Terrain Route Planning Agent

B.2.1 Code Calls

(1) Sets current map to `Map`
`route` : *useMap*(`Map`) → `Boolean`
(2) Compute a route plan on the current map for a vehicle of type `VehicleType` from `SourcePoint` to `DestPoint` given in 2D. Returns a route plan as a sequence of points in plane.
`route` : *getPlan*(`SourcePoint`, `DestPoint`, `VehicleType`)
→ `SequenceOf2DPoints`
(3) Given `SourcePoint` and `DestPoint` 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.
route : *groundPlan*(SourcePoint, DestPoint, VehicleType, Route)
→ (Route, Prob)

(4) Compute a flight plan on the current map from SourcePoint to DestPoint given in 3D. Returns a flight plan as a sequence of points in space
route : *flightPlan*(SourcePoint, DestPoint)
→ 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.
route : *visible*(Map, Point1, Point2) → Boolean

B.2.2 Actions

(1) Compute a route plan on map Map for a vehicle of type VehicleType from SourcePoint to DestPoint given in 2D.
Name: *planRoute*(Map, SourcePoint, DestPoint, VehicleType)
Pre: SourcePoint ≠ DestPoint
Del: {}
Add: **in**(**true**, route : *useMap*(Map))&
in(Plan, route : *getPlan*(SourcePoint, DestPoint, VehicleType))

(2) Given SourcePoint and DestPoint 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
Name: *evaluateGroundPlan*(Map, SourcePoint, DestPoint, VehicleType, Route)
Pre: SourcePoint ≠ DestPoint
Del: {}
Add: **in**(**true**, route : *useMap*(Map))&
in(RP, route : *groundPlan*(SourcePoint, DestPoint, VehicleType, Route))

(3) Compute a flight plan on map Map from SourcePoint to DestPoint given in 3D.
Name: *planFlight*(Map, SourcePoint, DestPoint)
Pre: SourcePoint ≠ DestPoint
Del: {}
Add: **in**(**true**, route : *useMap*(Map))&
in(Plan, route : *flightPlan*(SourcePoint, DestPoint))

B.3 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.3.1 Code Calls

(1) Get position for agent with id `AgentId`
`track` : *getPos*(`AgentId`) → `2DPoint`
(2) Get the type of agent for agent with id `AgentId`. It returns the most likely vehicle type together with the probability
`track` : *getTypeOfAgent*(`AgentId`) → (`VehicleType`, `Prob`)
(3) Return the list of all agents being tracked
`track` : *getListOfAgents*() → `ListOfAgentIdsF`

B.4 Coordination Agent

B.4.1 Code Calls

(1) Determine wether a vehicle of type `VehicleType1` at position `Pos1` can attack a vehicle of type `VehicleType2` at position `Pos2`. For example a tank is not able to attack a fighter plane unless it is on the ground.
`coord` : *canBeAttackedNow*((`VehicleType1`, `Pos1`, `VehicleType2`, `Pos2`)) → `Boolean`
(2) Given an agent id for an enemy vehicle, determine the best position, time and route for an attack to be successful. Return the estimated probability of success
`coord` : *findAttackTimeAndPos*(`AgentId`) → (`Pos`, `Time`, `Route`, `Prob`)
(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`
`coord` : *coordinatedAttack*((`SetOfAgentIds`, `EnemyId`)) → `AttackPlan`

B.4.2 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 partic-

ipating in the coordinated attack are taken from the set `SetOfAgentIds`.
Name: *attack*(`SetOfAgentIds`, `EnemyId`)
Pre: `SetOfAgentIds` $\neq \emptyset$
Del: {}
Add: **in**(`AP`, coord : *coordinatedAttack*((`SetOfAgentIds`, `EnemyId`)))

B.5 helicopter Agents

B.5.1 Code Calls

(1) Change flying altitude to `Altitude` (`0` to `Maximum` altitude)
`heli` : *setAltitude*(`Altitude`) → `Boolean`
(2) Get current altitude
`heli` : *getAltitude*() → `Altitude`
(3) Change flying speed to `Speed` (`0` to `Maximum` speed)
`heli` : *setSpeed*(`Speed`) → `Boolean`
(4) Get current speed
`heli` : *getSpeed*() → `Speed`
(5) Change flying heading to `Heading` (`0` to `360`)
`heli` : *setHeading*(`Heading`) → `Boolean`
(6) Get current heading
`heli` : *getHeading*() → `Heading`
(7) Aim the gun at the 3D point given by `Pos`
`heli` : *aim*(`Pos`) → `Boolean`
(8) Fire the gun using the current aim
`heli` : *fire*() → `Boolean`
(9) Determine the current position in space
`heli` : *getPos*() → `3DPoint`
(10) Compute heading to fly from 2D point `Src` to 2D point `Dst`
`heli` : *computeHeading*(`Src`, `Dst`) → `Heading`)
(11) Compute the distance between two 3D points
`heli` : *computeDistance*(`X`, `Y`) → `Distance`
(12) Retrieve the maximum range for the gun
`heli` : *getMaxGunRange*() → `Distance`
(13) Calculate the next position of the helicopter given its `CurrentPos`, its `Speed` and flying `Angle`
`heli` : *calculateNextPos*(`CurrentPos`, `Speed`) → `3DPoint`

B.5.2 Actions

(1) *Fly* from 3D point `From` to 3D point `To` at altitude `Altitude` and with speed `Speed`

Name: *fly*(From, To, Altitude, Speed)
Pre: **in**(From, heli : *getPos*())
Del: **in**(From, heli : *getPos*())
Add: **in**(CurrentPos, heli : *getPos*())&

in(To, heli : *calculateNextPos*(CurrentPos, Speed))

(2) *FlyRoute*(*path*) Path given as a sequence of quadraples consisting of: a 3D point, altitude, speed and angle
Name: *flyRoute*(Path)
Pre: **in**(Path(0).Pos, heli : *getPos*())
Del: **in**(Path(0).Pos, heli : *getPos*())
Add: **in**(Path(Path.Count).Pos, code call), where code call is

heli : *calculateNextPos*(Path(Path.Count − 1).Pos, Speed)

(3) Attack vehicle at position Pos in space from position MyPos
Name: *attack*(MyPos, Pos)
Pre: **in**(MyPos, heli : *getPos*())&

in(Distance, heli : *computeDistance*(MyPos, Pos))&

in(maxRange, heli : *getMaxGunRange*())&Distance < maxRange
Del:
Add:

B.5.3 Integrity Constraints

in(S, heli : *getSpeed*()) ↩ $S <$ maxSpeed
in(A, heli : *getAltitude*()) ↩ $A <$ maxAltitude

B.5.4 Action Constraints

{*fly_plane1*(X1, Y1, A1, S1), *fly_plane2*(X2, Y2, A2, S2) } ↩ Y1 =Y2

{*attack*(P)} ↩ **in**(P, heli1 : *getPos*())
{*attack*(P)} ↩ **in**(P, heli2 : *getPos*())
{*attack*(P)} ↩ **in**(P, heli3 : *getPos*())
{*attack*(P)} ↩ **in**(P, heli4 : *getPos*())

where heli1, ..., heli4 are the friendly agents of the agent in question.