technische universität dortmund

department of computer science

TECHNICAL REPORTS IN COMPUTER SCIENCE

Technische Universität Dortmund

Know-How for Motivated BDI Agents (Extended Version)

Matthias Thimm and Patrick Krümpelmann

Chair VI – Information Engineering

Number: 822
February 2009

Technische Universität Dortmund — Fakultät für Informatik
Otto-Hahn-Str. 14, 44227 Dortmund

Matthias Thimm and Patrick Krümpelmann: *Know-How for Motivated BDI Agents (Extended Version),* Technical Report, Faculty of Computer Science, Technische Universität Dortmund. © February 2009

ABSTRACT

The BDI model is well accepted as an architecture for representing and realizing rational agents. The beliefs in this model are focused on the representation of beliefs about the world and other agents and widely independent from the agents intentions. We argue that also the representation of *know-how*, which captures the beliefs about actions and procedures, has to be taken into account when modeling rational agents. Using the notion of *know-how* as introduced by Singh we formalized and implemented a usable and concrete agent architecture that benefits from this representation of procedural beliefs in multiple ways and also supports the representation of motivations that influence the agent's behavior. We enable the agent to reason about its planning capabilities in the same way as it can reason about any other of its beliefs by extending a BDI-based agent architecture to allow the representation of procedural beliefs explicitly as part of the agent's logical beliefs which again influences and enhances the agents behavior.

CONTENTS

1 INTRODUCTION

Planning and agency are two closely related fields in the research of multiagent systems and artificial intelligence in general. In this paper we take a look at the planning capabilities of an agent from the perspective of knowledge representation by explicitly modeling these capabilities as logical beliefs of the agent in order to enable it to reason about and revise them in the same way as any other logical beliefs. We build upon the commonly known and used BDI model [31, 24, 23] that divides the mental state of an agent into *Beliefs*, *Desires*, and *Intentions*. In [26] Singh introduced the notion of *know-how* in order to relate the components *Beliefs* and *Intentions* more closely. Know-how describes the part of the logical beliefs of an agent that describes structural knowledge to reach certain goals. While in most modern agent architectures [2] beliefs influence the selection and achievement of intentions the other way around is not supported by these systems in general. Singh [26] claims, that "*since rational agency is intimately related to actions and procedures, it is important also to consider the form of knowledge that is about actions and procedures*". In this paper we take a first step towards the support and implementation of structural knowledge about procedures, i. e. know-how, in a concrete agent architecture [30]. We show how the explicit representation of know-how in an agent can be realized and in which ways this representation gives rise to advantages in reasoning about the agent's beliefs. We support our formal work by an implementation of our know-how formalism in the full fledged BDI multiagent system KiMAS [15] which also supports motivations [19] that may influence goal adoption and the overall behavior of the agent.

This paper is organized as follows. In Section 2 we give some motivation and examples, followed by a short overview of the KiMAS agent system that extends the commonly known BDI model in several directions in Section 3. We continue with our formal proposal of know-how in Section 4 and illustrate the use of know-how in logic programming in Section 5. In Section 6 we review related work and in Section 7 we conclude.

2 BELIEFS AND KNOW-HOW

The BDI model [31, 23, 24] has become a leading paradigm of the research in agent representation and realization. This model distinguishes between *Beliefs*, *Desires*, and *Intentions* as the main elements of an agent's mind. Informally speaking, *Beliefs* comprise the agent's beliefs about the world and its current situation, *Desires* represent what it wishes to achieve and hence represent possible goals, whereas *Intentions* model the agent's immediate (sub-)goals and thus focus on the next actions the agent should undertake in order to achieve the current goal.

Know-how as introduced by Singh [26, 27] is an extension to the traditional BDI model. In [26] it has been argued that on a descriptive layer an explicit distinction between beliefs about the world (know-that) and beliefs about how to achieve certain intentions (know-how) is indispensable in order to model agents and their behavior in a natural way. But as know-how is a form of belief, an agent should be able to reason about these beliefs and to revise them. In modern formalizations of the BDI model or other agent architectures for planning and reasoning [3, 2, 17, 29, 25] planning and beliefs components are mostly kept separate. Although beliefs do (of course) influence intention deliberation and goal generation, the other way round, namely reasoning about the current intentions of the agent and especially reasoning about the capability of how to achieve some state cannot be formalized in a natural way in the above systems. Particularly, the pure knowledge of the possibility to achieve a certain intention is crucial for the agent in order to determine if the intention can be pursued or has to be dropped. In general, there are many situations in which it is necessary for the agent to reason about its planning capabilities.

Example 1. Imagine a cleaner robot that pursuits two goals, namely cleaning all rooms in his area and maintain a high battery level. It is crucial for the robot that it *knows how* to reach the charging station before beginning to clean the rooms at any time. Moreover, to efficiently do its task the robot should be able to consider if it can return to the charging station in time when planning more than one step ahead.

Example 2. Suppose two agents want to determine time and location for a meeting. The agents do not only have to consider their general capabilities of traveling, e. g. if they have a driving license, but also if their plans for traveling are applicable. If the only road between the first agent's location and the location of the meeting is closed due to road works, then it may not have the capability to reach the meeting in time.

Although the last example can be modeled in several existing agent architectures using only ordinary knowledge representation techniques, there is a difference between checking the beliefs for plan applicability and reasoning about the plans directly. The formalization of an agent becomes unclear and complicated when plan applicability has to be modeled in the beliefs of the agent (in

order to answer queries of other agents as in Example 2) as well as in the actual planning component. This also complicates the programming and maintenance of agents from an implementation point of view.
In [26] Singh develops a temporal logic in which know-that and know-how can be specified in order to verify whether an agent is able to achieve certain goals and whether it knows how to achieve them. In this paper, we take the general idea of know-how one step further on the way from a specification and verification tool towards a programming tool [30]. To our knowledge the concept of know-how has not been developed further since Singh's publications in the late 1990s while planning and intention generation are active fields [16, 4] including some ideas of how to combine state-of-the-art knowledge representation and reasoning techniques with problem solving and planning, e. g. [17]. Here we introduce know-how as a formalization of an agent's planning capabilities in a declarative manner and show how the agent can use this representation to reason about it.

Another problem with handling belief and planning components separately is revision of plans [15, 14, 13, 1]. Although existing agent architectures like Jason [3] allow rudimentary revision of plans in form of addition and deletion of plans, sophisticated revision techniques are usually not supported. When considering know-how as beliefs it is reasonable to apply standard belief revision techniques for this planning knowledge as well in order to benefit from this large body of work. Treating planning knowledge as beliefs lets the agent get revision techniques for planning knowledge for free as this representation enables the agent to revise this knowledge in the same manner as other beliefs. Revision of planning knowledge (and not just addition and deletion of fragments of structural knowledge) can be useful in many different circumstances.

Example 3. Consider again the cleaner robot from Example 1. Suppose one of the rooms the robot is ordered to clean has two doors, one of them is blocked and the other is passable. Suppose now that due to rearrangement of the furniture the first door is passable and the second one is blocked. The robot now has to revise its structural knowledge to use the first door in order to know how to access the room.

Besides revision, the above example introduces another important aspect of rational agency that is closely related to revision: Learning. A rational agent should be able to reason about success and failure of its plans and actions and acquire new (structural) knowledge through experience. When modeling this structural knowledge as beliefs, learning from experience and belief revision become even more closely related. In the above example the robot might not have been explicitly informed about the rearrangement of the furniture and it has to learn the changes from experience. When trying to enter the room for the first time it will notice that the action of entering the room cannot be applied due to the blocked door. This information can be used by an appropriate belief revision operator to revise the (structural) beliefs accordingly.

3 THE FORMAL MODEL

We denote *Des* as the set of all desires or possible goals an agent can have, where a desire or a possible goal is an atom that an agent wants to become true in its beliefs. An agent maintains a subset $D \subseteq Des$ of all possible goals, e. g. the cleaning robot of Example 1 has the desires $D = \{clean\_all, battery\_healthy\}$. Every agent has, at each moment, one selected goal that it currently pursuits, denoted $selected(D)$. Similarly, an agent maintains a set of abstract intentions $I \subseteq Int$ where *Int* denotes the set of all possible intentions. In general, the set I is represented as a stack and is used as one whenever needed. The type of representation used will be evident given the context. Intentions describe the aims of the agent that it currently pursuits in order to fulfill its selected goal. These are represented as atoms as well but are more concrete than desires. For clarity of presentation, we use only propositional goals and intentions. Every time an agent selects a new goal, this goal becomes its next intention. At any time the set I correlates directly to the current pursuited goal $selected(D)$ and contains the next subgoals the agent wants to become true in order to fulfill its current goal $selected(D)$. Some intentions can be directly fulfilled by performing an atomic action and are called *atomic intentions*.

3.1 THE AGENT MODEL

For simplicity of representation we abstract from a concrete communication protocol between agents here and introduce the set P of possible perceptions that is used to model both perceptions from the environment and communication acts from other agents. We refer to e. g. [15, 5] for the treatment of this issue in general.

An agent in our framework is modeled using two components: a data component and a functional component. The data component is a passive component that stores the whole mental state of the agent and contains its desires, motivations, capabilities, logical beliefs, intentions, and know-how. The last three are also subsumed by the notion of general "beliefs" as these can be manipulated in the same way as ordinary logical belief by the mechanisms introduced in this paper. To represent the logical beliefs of an agent we use extended logic programs as described in the previous section, so let $\mathcal{K}$ be the set of all extended logic programs. Let $\mathcal{C}$ be the set of all atomic actions an agent can perform, $\mathcal{KH}$ the set of all know-how bases, and *Mot* the set of all possible motivations (the last two will be explained later).

Definition 1 (Data Component). A *data component* is a tuple $A = (KB, \Sigma, D, I, C, M)$, where $KB \in \mathcal{K}$ is the logical belief base of the agent, $\Sigma \in \mathcal{KH}$ is the know-how base of the agent, $D \subseteq Des$, $I \subseteq Int$, $C \subseteq \mathcal{C}$ is a set of atomic actions the agent can perform, and $M \in Mot$ denotes the (basic) motivation of the agent.

The element Σ is the topic of this paper and will be described in more detail in the next section. Furthermore, we will shed some light on the element M in the next subsection.

Given an initial data component A, the agent acts and evolves in its environment and thus its data component changes over time. The procedures that determine how an agent uses its current state to deliberate its next actions are described by its *functional component*. The *functional component* of an agent is a set of functions revising its internal state in a particular situation or outputting an action that the agent wants to perform.

Definition 2 (Functional component). Let $\alpha_{bel}, \alpha_{upd}, \alpha_{subgoal}, \alpha_{select}, \alpha_{action}$ be functions defined as:

- $\alpha_{bel} : \mathcal{K} \times P \to \mathcal{K}$ (belief base operation)
- $\alpha_{upd} : \mathcal{K} \times \mathfrak{P}(Des) \times Mot \times \mathfrak{P}(Int) \to \mathfrak{P}(Int)$ (intention update operation)
- $\alpha_{subgoal} : \mathcal{K} \times \mathcal{KH} \times Int \to \mathfrak{P}(Int)$ (subgoal generation operation)
- $\alpha_{select} : \mathfrak{P}(Int) \times \mathcal{C} \to Int$ (intention selection operation)
- $\alpha_{action} : Int \to \mathcal{C}$ (action selection operation)

A tuple $\varphi = (\alpha_{bel}, \alpha_{upd}, \alpha_{subgoal}, \alpha_{select}, \alpha_{action})$ is called a *functional component*.

Definition 3 (Agent). An *agent* is a tuple (K, φ) with a data component K and a functional component φ.

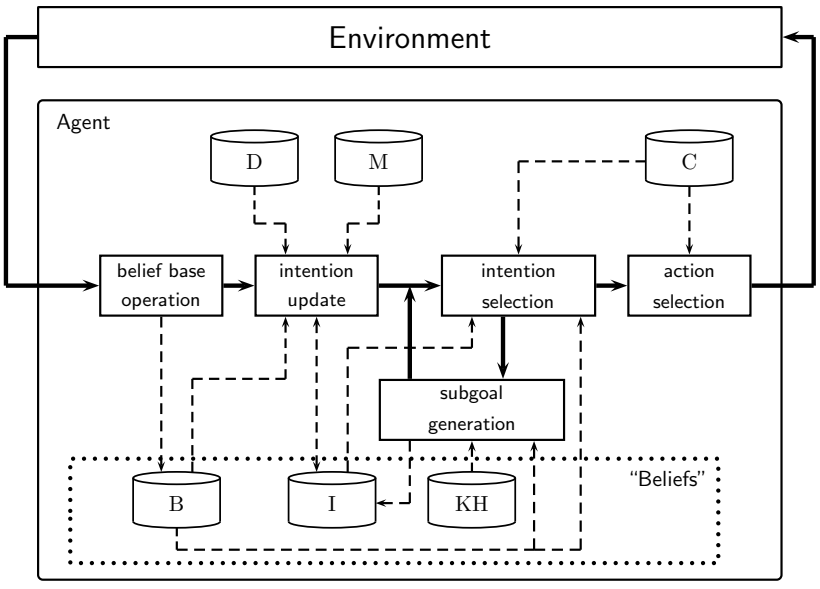

Figure 1: A graphical representation of the agent model. (The control flow is depicted with solid lines and the data flow with dashed lines.)

An agent is modeled as an infinite loop executing these operations as illustrated in Figure 1. When the agent receives a perception $p \in P$ from the environment, it revises its logical beliefs using its belief base operation yielding a new belief base. Following on this, the agent has to check whether its current intentions have to be updated or if a new goal has to be selected given the change of the world, using its intention update operation. The motivation M of the agent is used for the determination of a new selected goal as we will elaborate in Subsection 3.2. The intention selection operation then simply selects the top intention from the agent's intention stack and checks whether there is an atomic action to handle it. If that is the case, the action selection operation selects a suitable action (if there is more than one for the particular intention) and executes that action in the environment. If the intention cannot be fulfilled directly by means of an atomic action, it must be split up in less abstract intentions by using the agent's know-how. The subgoal generation operation uses the agents know-how to split up the current pursued goal or the next intentions into more concrete intentions. This operation is the topic of the next section. But before this, in order to motivate the whole agent model, we continue with some illustrations on two other components, namely motivation and the belief base operation.

3.2 MOTIVATIONS

The desires *Des* of autonomous agents correspond to the set of possible goals. An agent maintains a subset $D \subseteq Des$, but might not be able to aim at several of these desires simultaneously. Thus a mechanism is needed to decide which $d \in D$ will be taken into consideration next. We introduce motivations [19, 20] in this context. The possible motivations *Mot* are non-derivative components that characterise the personality of an autonomous agent and provide the agent with a higher-level control.

These components, e. g. greed or altruism, do not specify a state of affairs to be achieved and can hardly be described in logical terms. Therefore, they are not equal to the notion of goals in the classical sense of artificial intelligence. Instead motivations provide reasons for a goal, which could be having someone else's money or being generous. More precisely, in our system a motivation M induces a total preorder $\leq_M$ on the set of possible goals, which is a total, transitive and reflexive relation. Let $d_1, d_2 \in D$ be two desires, then $d_1 \leq_M d_2$ iff the motivation M for d_1 is at least as strong as for d_2. Thus the set of the agent's desires is partitioned into several sets $D = (D_0^M, \ldots, D_k^M)$ with

$$d_1 \in D_i^M \wedge d_2 \in D_j^M \wedge\ i \leq j \ \Leftrightarrow\ d_1 \leq_M d_2 \quad .$$

The next goal, $selected(D)$, will be a randomly selected $g \in D_0^M$ as the set D_0^M contains the desires which are motivated the most. To simplify matters we let every agent be driven by only one motivation M that forms the personality of the agent.

3.3 BELIEF BASE OPERATIONS

The environment of a multiagent system tends to be highly dynamic and in addition to that agents tend to interact with each other. These occurrences lead

to changes of the beliefs of an agent and are summarized as perceptions in this paper. On the arrival of new perceptions the belief base operation function $\alpha_{bel} : \mathcal{K} \times P \rightarrow \mathcal{K}$ incorporates the new information gained by the perception into the beliefs of the agent. This new information might be conflicting with former held beliefs of the agent as the environment might have changed or more specific information might be available. Therefore, the belief base operation function has to be capable of dealing with arising conflicts in a sensible manner in order to come to a consistent belief state and to enable reasoning based on this. Our framework can deal with a variety of belief base operators, potentially varying between different agents. The representation of the agents beliefs as extended logic programs enables the use of standard update mechanisms for these. In such a setting older or less accurate information has to be selected for rejection in order to come to a consistent belief set this rejection is widely based on preferences on rules [9]. Particularly interesting scenarios arise if the informations that are exchanged in the system are not completely trustworthy such that an appropriate processing of the diverse credibility of the available information has to be dealt with. For details on communication and metainformation handling in such a setting we refer to [15] and for details on credibility handling in multiagent systems to [14].

4 KNOW-HOW

The know-how of an agent is structured in *know-how statements* which is the atomic form of a structural piece of information.

Definition 4 (Know-How Statement). A *know-how statement* σ is a tuple

$$\sigma = (a, (s_1, \ldots, s_n), \{c_1, \ldots, c_m\})$$

with goals $a, s_1, \ldots, s_n \in Int$, $a \notin \{s_1, \ldots, s_n\}$ and literals $c_1, \ldots, c_m$. The goal a is called the *target*, the goals $s_1, \ldots, s_n$ are called the *sub-targets*, and the elements $c_1, \ldots, c_m$ are called the *conditions* of σ.

The informal meaning of a know-how statement

$$\sigma = (a, (s_1, \ldots, s_n), \{c_1, \ldots, c_m\})$$

is as follows: In order to achieve goal a the agent can try to achieve the subgoals $s_1, \ldots, s_m$, if the conditions $c_1, \ldots, c_m$ are fulfilled with respect to the agent's beliefs. Due to $a \notin \{s_1, \ldots, s_n\}$ we do not allow know-how statements to be recursive.

Example 4. We continue Example 1 of the cleaner robot. Suppose the robot has the top-level goal (desire) to clean all rooms in its area which is represented by *clean_all*. Suppose now that there are two rooms in its area, a hallway and a lounge, and that the robot shall verify that its battery is full before it starts cleaning these rooms. This knowledge can be captured by the know-how statement

$$\sigma = (cleaned\_all, (cleaned\_hallway, cleaned\_lounge), \{battery\_full\}).$$

The function $goal/1$ is used to denote the goal of a know-how statement, for example $goal(\sigma) = clean\_all$ for the know-how statement in Example 4.

Definition 5 (Know-How Base). A *know-how base* Σ is a finite set of know-how statements.

Example 5. We continue Example 4. Let Σ of the robot be given by $\Sigma = \{\sigma_1, \ldots, \sigma_5\}$:

$$\begin{aligned}
\sigma_1 &= (cleaned_all, (cleaned_hallway, cleaned_lounge), \\
&\quad \{battery_full\}) \\
\sigma_2 &= (cleaned_hallway, (at_hallway, \\
&\quad vacuumed_hallway), \{bag_empty\}) \\
\sigma_3 &= (cleaned_lounge, \\
&\quad (ordered_robotxy_to_clean_lounge), \\
&\quad \{robotxy_available\}) \\
\sigma_4 &= (cleaned_lounge, (at_lounge, \\
&\quad free_lounge, vacuumed_lounge), \{\}) \\
\sigma_5 &= (free_lounge, \\
&\quad (people_sent_away), \{at_lounge\})
\end{aligned}$$

The know-how statement σ_1 is taken from Example 4 and σ_2 states that in order to clean the hallway the robot has first to go to it and second to do the actual vacuuming. This statement can only be applied by the robot, if its vacuum cleaner bag is empty. The interpretation of the other statements should be clear by following this scheme. Observe that the robot has two alternatives for the intention *clean_lounge*: Given that the helper robot *robotxy* is present, our robot can order *robotxy* to do the job for it. Also note, that the fulfillment of the intention *at_lounge* in σ_4 is a prerequisite for the fulfillment of the intention *free_lounge* in σ_5. The intentions *at_hallway*, *at_lounge*, *vacuumed_hallway*, *vacuumed_lounge*, *ordered_robotxy_to_clean_lounge*, and *people_sent_away* are atomic intentions that can be fulfilled by executing the suitable atomic action, e. g. a walking action or an action to send away the inhabitants from the place of work.

In order to describe the semantics of know-how and the algorithm to use it we use state transition systems [21]. Besides the belief base and the know-how base, a record describing the current state of the plan deliberation is needed to fully describe an agent's mental state in our framework. Also to keep track of already failed plan deliberations we use a notion of intention tree similar to e. g. [28, 32] which generalizes the notion of an intention stack.

Definition 6 (Intention tree). An *intention tree* I is a tree with two sorts of nodes with the root node being a goal and for every node K it holds:

- iff K is a goal, then every child of K is a know-how statement σ with $goal(\sigma) = K$,
- iff K is a know-how statement σ, then the (ordered) children of K are the subgoals of σ.

Furthermore there is exactly one node, that is additionally labelled "*current*", which is also returned by the function $\mathsf{curr}(I)$. Furthermore the empty tree $I_\emptyset$ is also an intention tree with $\mathsf{curr}(I_\emptyset)$ being undefined.

Informally speaking, an intention tree I captures the current state of the pursuit of a goal. The initial state of an intention tree is described by a root node R that

is labelled with "*current*". For simplicity of presentation we assume that the agent only pursues one goal at a time, but the approach can be generalized by considering several intention trees. When a goal node K is labelled "*current*", then the agent selects a know-how statement with target K that is not already a child of that node and adds that know-how statement as a child of K. Here we assume that for every possible intention there is at least one know-how statement for that goal in the know-how base of the agent[1]. All sub-targets of that know-how statement are added as (ordered) children of that node and the "*current*" label is moved to the first sub-target. That process is repeated until an atomic goal is labelled with "*current*". Then the agent executes the corresponding action and the "*current*" label is moved to the next sibling (if it exists) or to the next sibling of the first know-how statement bottom-up that has not been fulfilled yet. Here, we assume that actions cannot fail and that they always bring about the intended results. When a know-how statement cannot be applied due to failure of the conditions or failure of some of the sub-targets, the agent selects another know-how statement for the corresponding target and goes on. If there is no know-how statement for a goal or all know-how statements failed, then the goal failed as well as its parent know-how node.

We now formalize the above intuition using state transition systems. An agent's mental state Ω is described by its data component $A = (KB, \Sigma, D, I, C, M)$ and an auxiliary label PREV-STATE

$$\Omega = \langle A, \textsc{prev-state} \rangle$$

The label PREV-STATE describes the output of the previous state of deliberation and is one of

- AP: an action has just been performed
- IA: an intention has been added to I
- KA: a know-how statement has been added to I
- KF: a know-how statement failed
- KP: a know-how statement has been completely processed
- NO: no operation has been performed

As the intention tree I is the only component that can be altered by the following state transition system, we will abbreviate the mental state

$$\langle (KB, \Sigma, D, I, C, M), \textsc{prev-state} \rangle$$

of an agent by $\langle I, \textsc{prev-state} \rangle$.

We will need some further notation for the upcoming state transition system. If I is an intention tree and A a node in I, then $\mathsf{children}(A)$ denotes the set of children of A. If A is labelled with an intention, then all nodes in $\mathsf{children}(A)$ are labelled with know-how statements and vice versa, due to Definition 6. If $\mathsf{curr}(I)$ is a goal and σ is a know-how statement, the function $\mathsf{addChild}(I, \sigma)$ returns a new intention tree I', that is a copy of I except that $\mathsf{curr}(I)$ get a new child σ and

1 See also the notion of *reliable know-how* in Section 5.2.1

the "*current*" label is moved to σ. Similarly, if $\mathsf{curr}(I)$ is a know-how statement and $s_1, \ldots, s_n$ are the sub-targets of σ, the function $\mathsf{addChildren}(I, \{s_1, \ldots, s_n\})$ returns a new intention tree I', that is a copy of I except that $\mathsf{curr}(I)$ gets new (ordered) children $s_1, \ldots, s_n$ and the "*current*" label is moved to s_1. For a goal node K the function $\mathsf{hasNext}(K, I)$ returns *true* if and only if K has a successor sibling in I given the ordering that was imposed on it and its siblings. Otherwise $\mathsf{hasNext}(K, I)$ returns $false$. Consecutively, if $\mathsf{hasNext}(K, I)$ is *true*, the function $\mathsf{nextSubtarget}(I)$ returns a new intention tree I', that is a copy of I except the "*current*" label is moved to the next sibling of K. For an arbitrary node K, the function $\mathsf{hasParent}(K, I)$ returns *true* if and only if there is an goal node $K' \neq K$ that is an ancestor of K in I. The function $\mathsf{toParent}(I)$ returns a new intention tree I', that is a copy of I except the "*current*" label is moved to the first intention node $K' \neq K$ that is an ancestor of K in I. For a know-how statement node K, the function $\mathsf{isApplicable}(K)$ returns *true* if and only if the conditions of K are valid conclusions of the agents knowledge base *KB*.

Now we are able to define a state transition system that describes the semantics of know-how. Given a certain mental state Ω of an agent, the next state Ω' is given by the following transition rules.

- Transition rules for state IA:

$$\frac{\mathsf{atomic}(\mathsf{curr}(I))}{\langle I, \mathsf{IA}\rangle \longrightarrow \langle I, \mathsf{AP}\rangle} \tag{1}$$

  Explanation: If the current intention of the agent is atomic, perform this action. The actual execution of the action and the integration of the result, i. e., new perceptions the agent might receive, are not incorporated in the above rule. As we are only interested in the planning behavior of the agent, this results in a transition from the intention-added state to the action-performed state.

$$\frac{\neg\mathsf{atomic}(\mathsf{curr}(I)) \wedge \sigma \in \Sigma \wedge goal(\sigma) = \mathsf{curr}(I)}{\langle I, \mathsf{IA}\rangle \longrightarrow \langle \mathsf{addChild}(I, \sigma), \mathsf{KA}\rangle} \tag{2}$$

  Explanation: If an intention has been added to be fulfilled and that intention is not atomic than add a know-how statement for this intention to the intention tree. Remember that we assume that there is at least on know-how statement for every intention in the know-how base of an agent.

- Transition rules for state AP:

$$\frac{\mathsf{hasNext}(\mathsf{curr}(I), I)}{\langle I, \mathsf{AP}\rangle \longrightarrow \langle \mathsf{nextSubtarget}(I), \mathsf{IA}\rangle} \tag{3}$$

  Explanation: If the current pursued intention has another sub-target after that one that has just been fulfilled, then set this sub-target as next intention.

$$\frac{\neg\mathsf{hasNext}(\mathsf{curr}(I), I) \wedge \mathsf{hasParent}(\mathsf{curr}(I), I)}{\langle I, \mathsf{AP}\rangle \longrightarrow \langle \mathsf{toParent}(I), \mathsf{KP}\rangle} \tag{4}$$

Explanation: If the just fulfilled intention was the last sub-target of the parent know-how statement, then this know-how statement has been completely performed.

$$\frac{\neg\mathsf{hasNext}(\mathsf{curr}(I), I) \wedge \neg\mathsf{hasParent}(\mathsf{curr}(I), I)}{\langle I, \mathsf{AP}\rangle \longrightarrow \langle I_\emptyset, \mathsf{NO}\rangle} \tag{5}$$

Explanation: If the just fulfilled intention was the last sub-target of the parent know-how statement and its parent intention is the root intention, i. e. the current goal, then the agent is finished and does nothing.

- Transition rules for state KP:

$$\frac{\neg\mathsf{hasNext}(\mathsf{curr}(I), I) \wedge \mathsf{hasParent}(\mathsf{curr}(I), I)}{\langle I, \mathsf{KP}\rangle \longrightarrow \langle \mathsf{toParent}(I), \mathsf{KP}\rangle} \tag{6}$$

Explanation: If a know-how statement has been performed and the parent intention of this statement was the last sub-target of its parent know-how-statement, then the current know-how statement has been performed.

$$\frac{\mathsf{hasNext}(\mathsf{curr}(I), I)}{\langle I, \mathsf{KP}\rangle \longrightarrow \langle \mathsf{nextSubtarget}(I), \mathsf{IA}\rangle} \tag{7}$$

Explanation: If the parent intention of the just performed know-how statement has a successor intention, then set that intention as next to be fulfilled.

$$\frac{\neg\mathsf{hasNext}(\mathsf{curr}(I), I) \wedge \neg\mathsf{hasParent}(\mathsf{curr}(I), I)}{\langle I, \mathsf{KP}\rangle \longrightarrow \langle I_\emptyset, \mathsf{NO}\rangle} \tag{8}$$

Explanation: If the parent intention of the just performed know-how statement is the root intention, i. e. the current goal, then the agent is finished and does nothing.

- Transition rules for state KA:

$$\frac{\mathsf{isApplicable}(\mathsf{curr}(I)) \wedge \mathsf{curr}(I) = (a, (s_1, \ldots, s_n), \{c_1, \ldots, c_m\}))}{\langle I, \mathsf{KA}\rangle \longrightarrow \langle \mathsf{addChildren}(I, (s_1, \ldots, s_n)), \mathsf{IA}\rangle} \tag{9}$$

Explanation: If the just selected know-how statement is applicable, then add all sub-targets to the intention tree and select the first sub-target as next intention.

$$\frac{\neg\mathsf{isApplicable}(\mathsf{curr}(I))}{\langle I, \mathsf{KA}\rangle \longrightarrow \langle \mathsf{toParent}(I), \mathsf{KF}\rangle} \tag{10}$$

Explanation: If the just selected know-how statement is not applicable, then it failed.

- Transition rules for state KF:

$$\frac{\sigma \in \Sigma \wedge goal(\sigma) = \mathsf{curr}(I) \wedge \sigma \notin \mathsf{children}(\mathsf{curr}(I))}{\langle I, \mathsf{KF}\rangle \longrightarrow \langle \mathsf{addChild}(I, \sigma), \mathsf{KA}\rangle} \tag{11}$$

Explanation: If there exists at least one other know-how statement with the same goal as the know-how statement that has just failed and this know-how statement has not already been applied, then add this know-how statement to the intention tree.

$$\frac{(\neg\exists\sigma \in \Sigma : goal(\sigma) = \mathsf{curr}(I) \wedge \sigma \notin \mathsf{children}(\mathsf{curr}(I))) \wedge \mathsf{hasParent}(\mathsf{curr}(I))}{\langle I, \mathsf{KF}\rangle \longrightarrow \langle \mathsf{toParent}(I), \mathsf{KF}\rangle} \tag{12}$$

Explanation: If all know-how statements for the current intention failed and the current intention is itself a sub-target of another know-how statement, then this know-how statement failed as well.

$$\frac{(\neg\exists\sigma \in \Sigma : goal(\sigma) = \mathsf{curr}(I) \wedge \sigma \notin \mathsf{children}(\mathsf{curr}(I))) \wedge \neg\mathsf{hasParent}(\mathsf{curr}(I))}{\langle I, \mathsf{KF}\rangle \longrightarrow \langle I_\emptyset, \mathsf{NO}\rangle} \tag{13}$$

Explanation: If all know-how statements for the current intention failed and the current intention is the root intention, i. e. the current goal, then the agent has failed completely and does nothing.

Figure 2 gives an overview of the whole transition system. Observe, that we do not give transition rules for the state NO. This is no accident but pure intent, as in this case, the motivation of the agent has to deliver a new goal the agent is going to pursuit. So addition of new goals is outside the responsibility of the above given transition system but must be done by external mechanisms. The above transition system is applicable when a new goal has been added by the agent's motivational component and thus starts in the state IA. Then the system tries to fulfill the goal by applying the transition rules an eventually stops in state NO. Furthermore, the outcome of the agent's actions are also not included in the transition system as this has to be done by the agent's belief base operation. But as we assume that actions cannot fail the transition rule (1) is valid for the purpose of describing the semantics of know-how.

We show now that the above transition system is sound and complete in the sense, that every run uniquely (given identical outcomes of the agent's actions) determines a sequence of states that eventually halts in state NO. In the following proofs, we assume the the component I of the agent cannot be altered outside the scope of the above transition systems. This means, that for example the perception component of the agent cannot modify the intention tree. There is one exception to this rule, as the motivational component of the agent can add new intentions if necessary. We neglect this case for the upcoming proofs because at this point we are interested only in the behavior of the planning component alone.

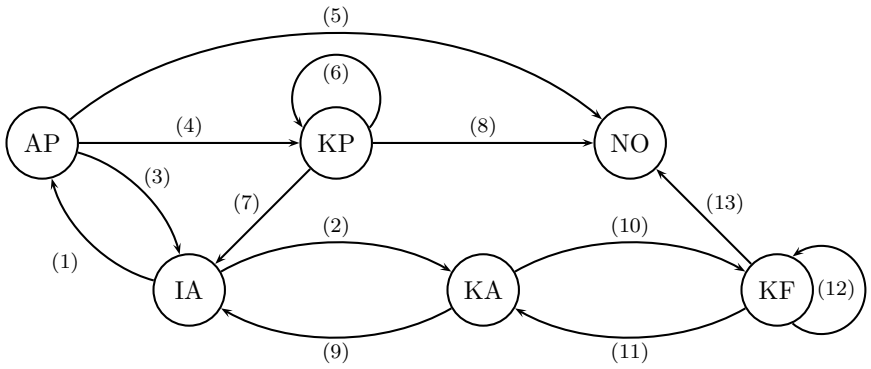

Figure 2: An overview over the transition system.

Proposition 1 (Soundness). The set of state transition rules (1)-(13) is sound in the sense, that for every mental state Ω of an agent, there is not more than one transition rule applicable.

Proof. By comparing the conditions of transition rules for the same state it can easily be seen, that these are mutually exclusive. □

Proposition 2 (Completeness). The set of state transition rules (1)-(13) is complete in the sense, that for every mental state Ω of an agent except a state with label NO, there is at least one transition rule applicable. Furthermore, every run of the system eventually halts in a state with label NO.

Proof. The execution of the transition system always starts in $\langle I, \mathsf{IA} \rangle$ with I being composed of one single intention, that has just been added by the motivational component of the agent. As one can easily see, every transition rule (1)-(13) generates output states that are valid input states for the next transition rules. For example, rule (11) creates a successor state where the "current" label is moved to a know-how statement which is expected by the transition rules for state KA (9) and (10). Furthermore, the conditions of the transition rules for the same state are exhaustive in the sense, that in every state at least one of them is applicable (except in NO). As the know-how base of the agent is finite and there are not recursive know-how statements (cf. Definition 4), there can be no infinite run of the system. As the system has eventually come to halt but every state except NO enables at least one following transition, the system will halt in NO. □

We now present an implementation of the functions α_{select}, α_{action}, $\alpha_{subgoal}$ from our agent model by one single algorithm, that realizes the above transition rules. This algorithm `NextAction` is depicted in figure 3 and takes an agent's mental state $\langle I, \text{PREV-STATE} \rangle$ and returns the next atomic action as output. Furthermore it modifies the agent's intentions according to its deliberations. The notation used in the algorithm `NextAction` is the same as used in the transition system.
As one can see, the algorithm `NextAction` implements exactly the behavior of the transition rules (1)-(13) and thus is also sound and complete. We illustrate

```
function NextAction

Input: a mental state ⟨I, prev-state⟩
Output: an atomic action A
Sideeffect: an updated mental state ⟨I, prev-state⟩


01  if prev-state = NO is empty then
02      return noop
03  if prev-state = IA then
04      if curr(I) is atomic then
05          I = I
06          prev-state = AP
07          return corresponding action A
08  else
10      let σ ∈ Σ with goal(σ) = curr(I)
11      I = addChild(I, σ)
12      prev-state = KA
13  else if prev-state = AP or prev-state = KP then
14      if hasNext(curr(I), I) then
15          I = nextSubtarget(I)
16          prev-state = IA
17  else if curr(I) is root intention then
18      I = I∅
19      prev-state = NO
20  else
21      I = toParent(I)
22      prev-state = KP
23  else if prev-state = KA
24      if curr(I) = (a, (s1, ..., sn), {c1, ..., cm}) is applicable then
25          I = addChildren(I, (s1, ..., sn))
26          prev-state = IA
27      else
28          I = toParent(I)
29          prev-state = KF
30  else if prev-state′ = KF
31      let F = {σ ∈ Σ | goal(σ) = curr(I) ∧ σ ∉ children(curr(I))}
32      if F ≠ ∅ then
33          let σ ∈ F
34          I = addChild(I, σ)
35          prev-state = KA
36      else if curr(I) is root intention then
37          I = I∅
38          prev-state = NO
39      else
40          I = toParent(I)
41          prev-state = KF
42  goto 01
```

Figure 3: The algorithm NextAction

the application of the algorithm `NextAction` using the example of the cleaner robot.

Example 6. Let *cleaned_all* be the initial intention of the agent, thus let I be composed of the single node *cleaned_all* and the initial mental state of the agent $A = \langle I, \mathsf{IA} \rangle$. Let now `NextAction` be called with A. As *cleaned_all* is not atomic, the lines 10 through 12 of algorithm `NextAction` are executed. Thus the know-how statement σ_1 is added to I and PREV-STATE is set to KA. Then the algorithm continues at line 01. Now it is PREV-STATE $=$ KA and suppose σ_1 is applicable. Then lines 25 and 26 of the algorithm are executed resulting in a new intention tree where the intentions *cleaned_hallway* and *cleaned_lounge* are added under σ_1 and the "*current*" label is moved to *cleaned_hallway*. Appropriately PREV-STATE is now set to IA and the agent selects the know-how statement σ_2 to satisfy *cleaned_hallway* yielding a new mental state with PREV-STATE $=$ KA. Suppose σ_2 is applicable, then the intentions *at_hallway* and *vacuumed_hallway* are added under σ_2 and the "*current*" label is moved to *at_hallway*. As now it is PREV-STATE $=$ IA and *at_hallway* is atomic, the lines 05 through 07 are executed. The intention tree of the agent is preserved and PREV-STATE is set to AP. The action is returned by the algorithm and executed in the environment of the agent. After incorporating new perceptions the agent continues with another call of `NextAction` and the preserved mental state from the last execution. As there is another sub-target of σ_2 the lines 15 and 16 of the algorithm are executed. The "*current*" label is moved to *vacuumed_hallway* and PREV-STATE is set to IA. Now lines 05 through 07 are applicable and the agent executes the action corresponding to the atomic intention *vacuumed_hallway*. After incorporating new perceptions, the agent calls again `NextAction`. Now it is PREV-STATE $=$ AP, there is no other sub-target of the know-how statement σ_2 to fulfill, and $\mathsf{curr}(I) =$ *vacuumed_hallway* is not the root intention, so lines 21 and 22 are executed. The "*current*" label is moved to *cleaned_hallway* and PREV-STATE is set to KP. As there is another sub-target of σ_1 lines 15 through 16 are executed. Now the "*current*" label moves to *cleaned_lounge* and PREV-STATE is set to IA. As *cleaned_lounge* is not atomic, the lines 10 through 12 are executed. As there is more than one know-how statement for *cleaned_lounge* let σ_3 be chosen by the agent to fulfill *cleaned_lounged*. So σ_3 is added to the intention tree under *cleaned_lounge* and the "*current*" label is moved onto it. Suppose now that σ_3 is not applicable, thus resulting in the execution of lines 28 and 29. The "*current*" label is moved back on *cleaned_lounge* at PREV-STATE is set to KF. As there is another know-how statement for *cleaned_lounge* lines 33 through 35 are executed. Now σ_4 is added under *cleaned_lounge* and PREV-STATE is set to KA. Suppose now that all remaining know-how statements are applicable. Then the agent consecutively performs the corresponding actions of the atomic intentions *at_lounge*, *people_sent_away*, and *vacuumed_lounge*, eventually leading to a mental state where PREV-STATE $=$ AP and the "*current*" label of I is on *vacuumed_lounge*. Now lines 21 and 22 are executed twice as both σ_5 and σ_4 were completely performed and thus are their parent intentions. Now the "*current*" label is moved to the intention *cleaned_all* and PREV-STATE is set to KP. As *cleaned_all* is the root intention lines 18 and 19 are executed yielding to the final mental state of the agent with an empty intention tree and PREV-STATE $=$ NO. At last line 02 is executed and the agent returns the empty action. Every fol-

lowing call of `NextAction` will return the empty action until the motivational component of the agent adds a new root intention to the intention tree.

The algorithm `NextAction` is similar to many existing planning algorithms, e. g., [28, 32]. In the next section we will present an implementation of that algorithm and a representation of know-how in logic programming. With this representation the agent is able not only to use its know-how for means-end reasoning, but also to reason about it for arbitrary purposes, which is not possible in other approaches for planning.

5 REASONING, APPLICATIONS, AND IMPLEMENTATION

For what is coming, we use extended logic programs under the answer set semantics [11]. These are capable of dealing with incomplete information in open environments. An extended logic program consists of rules over a set of atoms $\mathcal{L}$ using strong negation $\neg$ and default negation not. Rules are of the form $H(r) \leftarrow \mathcal{B}^+, \mathcal{B}^-$. where the body is devided into a positive, $\mathcal{B}^+$, and a negative, $\mathcal{B}^+$, set of literals in terms of default negation. P^S denotes the reduct of a program P relative to a set S of literals and is defined as: $P^S := \{H(r) \leftarrow \mathcal{B}^+(r) | r \in P, \mathcal{B}^-(r) \cap S = \emptyset\}$. An answer set of a program P is an interpretation I which is a minimal model of P^I.

Here, we extend the syntax of extended logic programs with lists as in Prolog. A *list* is a sequence of literals enclosed by squared brackets, e. g. $[a, b, c]$, that can appear as arguments for predicates. We use the notion $[H \mid B]$ to divide the list in the first element H and the remaining list B. This is a shorthand to simplify presentation and does not extend the answer set semantics. In fact, under some conditions (that are fulfilled in the context described here) it can be shown, that extended logic programs with lists can be rewritten in extended logic programs without lists, so that the answer sets of the rewritten program can be appropriately rewritten to answer sets with lists [18]. A yet simpler treatment of lists in answer set programming that is sufficient for our needs here is given in Appendix A.1.

5.1 KNOW-HOW IN LOGIC PROGRAMMING

Firstly we translate the know-how base of an agent into a logic program, where all know-how statements are represented as facts.

Definition 7 (Induced LP Know-How Statement). Let $\sigma = (a, (s_1, \ldots, s_n), \{c_1, \ldots, c_m\})$ be a know-how statement. The *induced LP know-how statement* σ^L is the logic program:

$$
\begin{aligned}
&khStatement(\sigma, a).\\
&khSubgoal(\sigma, 1, s_1). \ \ldots \ khSubgoal(\sigma, n, s_n).\\
&khCondition(\sigma, c_1). \ \ldots \ khCondition(\sigma, c_m).
\end{aligned}
$$

Besides the induced LP know-how statements an induced LP know-how base also contains information about the atomicity of intentions, which is captured by facts using the predicate *is_atomic*.

Definition 8 (Induced LP Know-How Base). Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a know-how base. The *induced LP know-how base* Σ^L is defined as

$$\Sigma^L = \bigcup_{i=1}^{k} \sigma_i^L \cup \{is\_atomic(int). \mid int \text{ is atomic}\}.$$

In order to use the logical beliefs of the agent to check for applicability of know-how statements a translation mechanism is needed. The conditions of know-how statements are modeled as parameters in the facts of the induced LP know-how statements while the logical beliefs of the agent are modeled with literals. For example, the induced LP know-how statement of σ_1 from Example 5 yields besides others the logical fact $khCondition(\sigma_1, battery\_full)$, while the logical beliefs of the agent may contain the fact $BatteryFull$. In order to use the logical beliefs, we assume that for each literal L of arity zero appearing in the logical belief of the agent, there is also the rule

$$holds(l') \leftarrow L.$$

in the logical beliefs. There, l' is a new constant, that shall be identified with the literal L. For example, for the literal $BatteryFull$ we add the rule

$$holds(battery\_full) \leftarrow BatteryFull.$$

to the agents beliefs. As said, we only introduce these rules for literals of arity zero. For literals with arity greater zero these mechanism can be extended in order to allow conditions as well as intentions and goals to be parametrized. But to stay simple in our presentation we omit this extended mechanism and assume that all conditions of know-how statements can only appear as propositional literals in the agent's beliefs.

Given the above representation of know-how in logic programming we continue by presenting an implementation of the algorithm `NextAction` (see Figure 3) in logic programming. For this reason we need to represent the intention tree of the agent explicitly as facts in logic programming. This is done in the logic program I^L which represents the component I and contains at all times facts of the following predicates:

- $istack(is)$: is captures the intention stack of the agent as a list, the first intention being the currently pursued intention.
- $khstate(ks)$: ks represents the current stack of know-how statements corresponding to the intention stack.
- $act(ai)$: if the agent determined an atomic intention to be executed in the previous state of plan deliberation, this fact is added to I^L.
- $khFailed(kf)$: kf is a list indicating, that the first know-statement failed in the context of the rest of kf due to unfulfilled conditions.
- $khPerformed(kh)$: this fact is added to I^L if all sub-targets of the specified know-how statement were fulfilled in the previous state of plan deliberation.
- $transState(state)$: this fact describes the current state of plan deliberation by summarizing the last operation. Accordingly, *state* is one of
  - *actionPerformed* (an action has been performed)
  - *intentionAdded* (an intention has been added to *istack*)
  - *khAdded* (a know-how statement has been added to *khstate*)

– *khFailed* (a know-how statement failed)
– *khPerformed* (a know-how statement has been completely processed)
– *noop* (no operation has been performed)

The facts *istack*, *khstate* and *khFailed* altogether form an intention tree as described earlier. Initially the logic program I^L has the structure

$$
\begin{aligned}
&istack([initial\_intention]).\\
&khstate([]).\\
&transState(intentionAdded).
\end{aligned}
$$

with a given initial intention *initial_intention*. Given an induced LP know-how base Σ^L and the logic program I^L, we need a set of logic rules that uses these beliefs to generate the new state I^L in an answer set. These logic rules for computing the next action of the agent are given as follows.

- Rules for State *intentionAdded*:

$$
\begin{aligned}
(r_1)\quad & new\_act(A) \leftarrow istack([A|H]),\\
& \quad is\_atomic(A),\\
& \quad transState(intentionAdded).\\
(r_2)\quad & new\_khstate([KH|K]) \leftarrow khstate(K),\\
& \quad khStatement(KH, I),\\
& \quad istack([I|\_]),\\
& \quad khCondition(K, X),\\
& \quad \text{not } holds(X),\\
& \quad \text{not } kh\_failed([KH|K]),\\
& \quad transState(intentionAdded),\\
& \quad \text{not } new\_act(\_).\\
(r_3)\quad & new\_khFailed([KH|K]) \leftarrow transState(intentionAdded),\\
& \quad \text{not } new\_act(\_),\\
& \quad \text{not } new\_khState(\_),\\
& \quad khState([KH|K]).\\
(r_4)\quad & toParent \leftarrow new\_khFailed(\_).
\end{aligned}
$$

- Rules for State *actionPerformed*:

$$(r_5)\quad new\_istack([C|B]) \leftarrow khstate([KH|\_]),\ transState(actionPerformed),\ istack([A|B]),\ khSubgoal(KH, I, A),\ J = I + 1,\ khSubgoal(KH, J, C).$$

$$(r_6)\quad new\_khPerformed(KH) \leftarrow khstate([KH|\_]),\ transState(\ actionPerformed),\ istack([A|\_]),\ khSubgoal(KH, I, A),\ J = I + 1,\ \textsf{not}\ khSubgoal(KH, J, \_).$$

$$(r_7)\quad toParent \leftarrow new\_khPerformed(\_).$$

- Rules for State *khAdded*:

$$(r_8)\quad new\_istack([I|B]) \leftarrow istack(B),\ khSubgoal(KH, 1, I),\ transState(khAdded),\ khstate([KH|K]).$$

- State transistion rules:

$$(r_9)\quad new\_transState(actionPerformed) \leftarrow new\_act(A).$$

$$(r_{10})\quad new\_transState(intentionAdded) \leftarrow new\_istack(\_),\ \textsf{not}\ khPerformed(\_).$$

$$(r_{11})\quad new\_transState(khAdded) \leftarrow new\_khState(\_),\ \textsf{not}\ khPerformed(\_).$$

$$(r_{12})\quad new\_transState(intentionAdded) \leftarrow new\_khFailed(\_),\ \textsf{not}\ istack([]).$$

$$(r_{13})\quad new\_transState(noop) \leftarrow new\_istack([]).$$

- Auxiliary rules:

$$(r_{14})\quad new\_istack(B) \leftarrow toParent,\ istack([\_|B]).$$

$$(r_{15})\quad new\_khState(K) \leftarrow toParent,\ khState([KH|K]).$$

The rules above have a clear resemblance to the transition rules used to describe the semantics of know-how in Section 4. Let $\texttt{NextAction}^L$ be the logic program that contains rules (r_1) to (r_{15}). Using $\texttt{NextAction}^L$ we can state an algorithm that computes the next action for the agent, see Figure 4. This algorithm has a one to one correspondence to algorithm NextAction (see Figure 3) but uses answer set programming as representation language.

```
do
        Compute the intersection ans of all
                answer sets of I^L ∪ Σ^L ∪ KB∪NextAction^L
        Adjust I^L according to ans
until I^L contains a fact act(A)
execute the action corresponding to A.
```

Figure 4: The algorithm for plan deliberation

It repeatedly computes the intersection of all answer sets of the union of I^L, the know-how base, the logical belief base and the rules in $\texttt{NextAction}^L$. Due to the single instantiation of $transState(\cdot)$ in I^L the applicability of derivation of literals of the form $new\_action(\dots)$, $new\_istack(\dots)$, $new\_khstate$, etc. is the same for all possible answer sets (given that the logical beliefs in KB do not influence this derivation in an undesired manner), such that these literals are the same in every answer set. Finally, these literals are extracted from this intersection, their "$new\_$" prefixes are stripped off and they are set as the new program I^L.

Proposition 3 (Equivalence to the state transition system). The logic program $\texttt{NextAction}^L$ which consists of the rules $(r_1) - (r_{15})$ is equivalent to the state transition system described in Section 4 in the sense that for every transition state the same successor state with the same effects is reached. In particular for every mental state Ω of an agent except a state with label NO (*noop*), there will be a successor state reached within a finite number of steps. Furthermore, every run of the system eventually halts in a state with label NO (*noop*).

Proof. For the state *intentionAdded* one of the rules $(r_1) - (r_3)$ is applicable. These are mutual exclusive as the heads occur in the negative body of the other rules. Rule (r_1) will directly trigger (r_9) and resembles the behaviour of transition rule (1). (r_2) resembles (2), but does also check for the applicability of the know-how statement as will be explained later. (r_2) triggers (r_{11}) which conducts the state change as desired. The failure of the check for applicability of know-how statements in (r_2) will lead to the application of (r_3), if (r_1) is not applicable. At this point no applicable know-how statement for the current intention is available and therefore the current know-how statement failed (r_3). This triggers (r_4) which resembles the *toParent* operation of the state transition system by means of rules (r_{14}) and (r_{15}). At last (r_{12}) is applicable in the same iteration and marks the successor state to be *intentionAdded*. Thus, in the next iteration the applicability of (r_2) is checked again on the parent intention of the

intention that just proved to be not achievable. Also the know-how statement which just failed has been marked as failed and is not taken into consideration again. The described application of rules (r_3), (r_4), (r_{14}), (r_{15}) and (r_{12}) initiates a loop if (r_3) is applicable again. This loop is equivalent to the application of state transition rules (2), (10) and (12). The loop is terminated if (r_2) is finally applicable, which resembles (11), or if the intention stack is empty and (r_{13}) is applicable, which again resembles (13).

For the state *actionPerformed* one of the rules $(r_5) - (r_6)$ can be applicable which are mutual exclusive. Rule (r_5) resembles the state transition rule (3). Here, the next sub-target is realised by using the ordered structure of the $khSubgoal(\cdot,\cdot,\cdot)$ predicate. (r_6) plus (r_7) resemble (4) while (5) is captured by (r_{13}).

For the state *khAdded* r_8 will be applied and adds the first subgoal of the added know-how statement to the intention stack. Taking into consideration that the other subgoals are captured by the $khSubgoal(\cdot,\cdot,\cdot)$ predicate and that they will be added by means of the (r_5), (r_8) captures the behaviour of state transition rule (9). Finally, (10) for the state *khAdded* is covered by the applicability checks in (r_2) and (r_3). □

Corollary 1 (Soundness of the logic program representation). The logic program $\texttt{NextAction}^L$ which consists of the rules $(r_1) - (r_{15})$ is sound in the sense, that for every mental state Ω of an agent, there is not more than one new state defined by the presence of a predicate $new\_transState(\cdot)$ in the answer sets.

Corollary 2 (Completeness of the logic program representation). The logic program $\texttt{NextAction}^L$ which consists of the rules (1)-(15) is complete in the sense, that for every mental state Ω of an agent except a state with label NO (*noop*), there will be a successor state reached within a finite number of steps. Furthermore, every run of the system eventually halts in a state with label NO (*noop*).

5.2 REASONING ABOUT KNOW-HOW

One of the main motivations for the explicit representation of know-how and intentions in the agents beliefs is that this enables the agent to reason about these. The agents awareness about the procedural knowledge it has to achieve goals can deeply influence the behaviour of what we call its functional component. To be more precise, the information reflected by the know-how is useful for the selection of new goals as the achievement of these is critically dependent on know-how. The structure of know-how also reveals information on the involved conditions and subgoals of the achievement of goals. This information enables the agent to reason about the feasibility and effort as well of the reliability of the achievement of goals. The just described way of reasoning interacts with the agents motivations as these can be dependent on the current level of confidence of the agent in a given situation. The level of confidence can be determined by the applicability of the agent's know-how, weighted with the significance of its goals. These ideas lead to the definition of sound and reliable know-how in the following subsections.

$$
\begin{array}{ll}
achievable(I) & \leftarrow is\_atomic(I). \\
achievable(I) & \leftarrow khStatement(KH, I), \\
 & \quad \text{not } \neg sound(KH). \\
\neg sound(KH) & \leftarrow khSubgoal(KH, J, SI), \\
 & \quad \text{not } achievable(SI).
\end{array}
$$

Figure 5: Determination of sound know-how

5.2.1 *Sound Know-How*

Example 2 showed that an agent must be capable of determining if it has the means to achieve a given intention. This is captured by the following definition.

Definition 9 (Sound Know-How). Let Σ be a know-how base. An intention $I \in Int$ is *achievable* in Σ if

- I is an atomic intention or
- there is at least one know-how statement $\sigma \in \Sigma$ with $goal(\sigma) = I$ and every sub-target of σ is achievable.

Σ is called *sound* if every intention $I \in Int$ is achievable.

Given an know-how base Σ^L in logic programming the rules stated in Figure 5 determine whether an intention I is achievable. The definition of sound know-how and the corresponding logic program do not take the conditions of a know-how statement into consideration when determining if an intention is achievable. In order to comprehend for these conditions we discuss in the next subsection a notion called *reliable know-how*.

5.2.2 *Reliable Know-How*

Singh states that reliable know-how should meet some form of natural language understanding and defines a very strong notion of reliable know-how in [26]. He also notes that alternative, less strict, versions of reliability might be formulated. A similar notion is the one of *secure planning* [8], which captures the intuition of a know-how statement or an intention, that can always be fulfilled no matter what the circumstances are. Due to uncertainty of the agent's beliefs, one might be interested if the agent is able to fulfill a given intention even if its beliefs are incomplete or if the environment changes due to other agents' actions. In our framework reliability of know-how is understood as the robustness of know-how to the incompleteness of information. Reliable know-how is know-how which is known not to fail given the incomplete information available. In particular this means that literals which are neither known to be true nor false are irrelevant for the success of the application of the know-how. As said above, we assume that actions cannot fail, so atomic intentions are *reliably achievable* by definition. For complex intentions *reliability* is recursively defined using the reliability of its subcomponents and a *context* as defined in the following.

$$
\begin{array}{ll}
reliablyAchievable(I, C) & \leftarrow is\_atomic(I). \\
reliablyAchievable(I, C) & \leftarrow khStatement(KH, I), \\
 & \quad \text{not } \neg reliable(KH, C). \\
\neg reliable(KH, C) & \leftarrow khCondition(KH, Cond), \\
 & \quad notin(Cond, C). \\
notin(X, []). & \\
notin(X, [H|B]) & \leftarrow X \neq H, notin(X, B). \\
\neg reliable(KH, C) & \leftarrow khSubgoal(KH, I, Int), \\
 & \quad context(C2, C, KH, I), \\
 & \quad \text{not } reliablyAchievable(Int, C2). \\
context(C, C, KH, 1). & \\
context([Int|C2], C, KH, I) & \leftarrow I \neq 1, \\
 & \quad J = I - 1, \\
 & \quad context(C2, C, KH, J), \\
 & \quad khSubgoal(KH, J, Int).
\end{array}
$$

Figure 6: Determination of reliable know-how

Definition 10 (Context). Let Σ be a know-how base and $\mathfrak{C}$ the set of all conditions of know-how statements σ with $\sigma \in \Sigma$. A *context* C is any subset of $\mathfrak{C}$.

Definition 11 (Reliability). Let Σ be a know-how base, I an intention and C a context. The intention I is *reliably achievable in* C iff

- I is an atomic intention or
- there is at least one know-how statement σ with target I and σ is reliable in context C.

A know-how statement σ with sub-targets $I'_1, \ldots, I'_n$ is *reliable in context* C iff

1. each sub-target I'_i $(1 \leq i \leq n)$ of σ is reliably achievable in $C \cup \{I_1, \ldots, I_{i-1}\}$ and
2. the conditions of σ are fulfilled in C.

An intention that is reliably achievable in the context $\emptyset$ is *absolutely reliably achievable*, as there are no necessary conditions for this intention to succeed. Note, that as we assume that actions do not fail and thus complex intentions do neither given they are reliably achievable, the intentions of previous sub-targets are added to the context of a sub-target as well, because previous actions may presuppose the application of later intentions.

Our notion of reliable know-how is implemented through the addition of the rules depicted in Figure 6. The rules in Figure 6 enable agents to always have knowledge about which of their know-how is reliable and which is not. This knowledge influence the intention update operation α_{upd} as described in Section 3.1. Thus, the knowledge about the reliability of know-how influences the behaviour of the agent as it can take care of some tasks as long as it reliably knows how to achieve some important goals. In the example of the cleaning robot this means, that it keeps on cleaning as long as it reliably knows how to get to the charging station. In general the motivation of agents is dependent

on the level of confidence of their current situation. While the motivation to achieve higher order goals while having a high sense of confidence the motivation to survive takes over in more unstable situations. This behaviour can be modelled by different preference relations on the agents motivations each reflecting a state of confidence. These states are based on the reliability of the know-how of the agent. Thereby the beliefs of the agent and the reasoning about these do influence other components of the agent such as its motivation.

5.3 BELIEF REVISION AND KNOW-HOW

In Section 3.3 we introduced the role of belief operations in our framework. Belief dynamics are usually limited to factual knowledge, referred to as know-that here. One of the main features of our approach of explicit representation of know-how is that this way know-how can as well be subject to dynamic changes carried out by belief operations. Especially, new know-how can be incorporated into existing know-how bases while maintaining consistency. Furthermore, the dynamics of classic beliefs, i. e. know-that, and of know-how are processed together respecting interactions of both. Updates of subgoals or conditions of know-how statements are automatically achieved if new subgoals or conditions are added for a given know-how statement and lead to a conflict. This conflict can either be a direct one which is handled by the update mechanisms directly or induced by integrity constraints of the form:

$$\leftarrow khSubgoal(KH, I, Int1), khSubgoal(KH, I, Int2), Int1 \neq Int2.$$

Updates of know-that will instantaneously lead to changes to the reliability of know-how which again will influence other components as laid out earlier. Another aspect in this manner is the communication and transfer of know-how [15]. Agents can ask other agents how to achieve certain intentions and as know-how is represented as beliefs, the answering of these queries can be handled analogue to other beliefs in a straightforward fashion. Agents receiving new know-how can integrate this new know-how into their own know-how base using ordinary belief base operations as described above.

5.4 IMPLEMENTATION

The system described in this paper has been implemented in the KiMAS framework that has been introduced in a previous paper [15]. The KiMAS framework is a multiagent system, that focuses on the representation of beliefs and on methods to process information exchanged among the agents. It has been implemented using the Jadex framework [22] and DLV [7] as the underlying answer set reasoner. While in the first version, a variant of the `NextAction` algorithm has been implemented in Java, we also implemented the logic programming version described in this section to fully benefit from the representation of intentions and know-how in logic programming.

6 RELATED WORK

There exist several approaches for plan generation, the most commonly known probably being the STRIPS language [10]. STRIPS is a declarative programming language that enables an agent to do means-end reasoning by stating atomic actions, featuring some conditions. Given a certain goal the STRIPS algorithm generates a sequence of atomic actions that lead to the fulfillment of this goal. With this paper we do not propose an alternative method for planning, but an orthogonal one. By using pre-written plan fragments to reach a certain goal, we do not fully have the means for plan deliberation. In fact, a planning algorithm like STRIPS can provide the input for building up a know-how base.

There is also a wide range of agent architectures that built on the BDI model [2, 3, 22]. All these architectures do not inherently support the notion of know-how nor reasoning about intentions and plan deliberation in the way described here. The system Jason [3], although, allows a simple treatment of revision of plans in the sense, that plan fragments may be added or deleted from the agent program. Nonetheless, we believe that our proposal might by adapted for several agent architectures in order to enrich them with a similar notion of know-how.

Extended logic programming has been extensively used as a language for plan deliberation. For example, in [17] Lifschitz uses extended logic programming in order to to provide solutions for the blocks world problem. In contrast to [17], our use of extended logic programming is much more general, as our algorithm is universal enough to handle a wide range of problems given a suitable know-how base. The system $\mathcal{K}$ [8] also makes use of extended logic programs as representation technique and also features a notion of *reliability* called *secure plans*. But like in other agent systems, $\mathcal{K}$ does not allow the treatment of structural knowledge about planning capabilities as ordinary logical beliefs.

7 CONCLUSION

In this paper we took on the work of Singh on the formalization and representation of know-how and focused on the realization of rational agents that are capable of the representation of and reasoning about their know-how. We formalized an extension of the BDI model for this sake, also including motivations for agents and set know-how apart from common planing approaches by representing know-how as logical beliefs in the mental state of the agent. By doing so, the agent acquires the capability to reason about its current state of plan deliberation within its logical beliefs and enables it to treat this kind of beliefs in the same way as its other beliefs. We presented a realization of know-how and its treatment in logic programming and illustrated the advantages that come with this representation. We see this proposal as a first step to the full support of the notion of know-how [26] in a concrete logic-based agent architecture. This constitutes an enhancement of the agents reasoning capabilities as well as it improves the interplay of the agents components. For future work, we plan to exploit and extend the new possibilities opened by our work in terms of reasoning with and about know-how as well as the further integration of agent components. Also, the notion of motivation has to be elaborated further in order to implement it in an entirely logic-based and practically usable agent architecture as was done here with know-how.

APPENDIX

A.1 USING LISTS IN ANSWER SET PROGRAMMING

The handling of lists is a key feature in the logic programming language PROLOG [6]. In this section, a very restricted form of lists, namely finite non-nested lists where every member does not appear more than once, are incorporated into extended logic programs under the answer set semantics [12]. The approach for integrating lists in extended logic programs presented here is very simple and straightforward. A more sophisticated approach can be found in [18]. There, a general handling of functions in normal logic program is presented. As lists can be represented using functions this approach is also applicable for the handling of lists. The approach also generalizes easy to extended logic programs.

A.1.1 *Syntax*

Let $\mathcal{C}$ be a finite set of constant symbols and $\mathcal{X}$ resp. $\mathcal{X}_L$ be two disjoint finite sets of variable symbols. Variables in $\mathcal{X}$ are called *c-term variables* and stand for the constants appearing in $\mathcal{C}$, while variables in $\mathcal{X}_L$ are called *list variables* and stand for lists (see below). By convention, constant symbols begin with a lowercase letter, while variable symbols begin with an uppercase letter. Here, we only consider a very restricted form of lists, namely lists that do not have lists as members and every element does not appear more than once.

Definition 12 (List). The set of *lists* $\mathbf{L}_{\mathcal{C},\mathcal{X},\mathcal{X}_L}$ for a set of constants $\mathcal{C}$ and disjoint sets of variables $\mathcal{X}$ and $\mathcal{X}_L$ is recursively defined as follows

- It is $[] \in \mathbf{L}_{\mathcal{C},\mathcal{X},\mathcal{X}_L}$ (the empty list).
- It is $\mathcal{X}_L \subseteq \mathbf{L}_{\mathcal{C},\mathcal{X},\mathcal{X}_L}$ (list variables).
- If $L \in \mathbf{L}_{\mathcal{C},\mathcal{X},\mathcal{X}_L}$ and $t \in \mathcal{C} \cup \mathcal{X}$ does not appear in L, then it is $[t|L] \in \mathbf{L}_{\mathcal{C},\mathcal{X},\mathcal{X}_L}$.

A list $[a_1|[a_2|\ldots[a_n|[]]\ldots]$ can also be written as $[a_1,\ldots,a_n]$ which is called the *normalized form*.

Remark 1. For a finite set of constant symbols $\mathcal{C}$ and finite sets of variable symbols $\mathcal{X}$ and $\mathcal{X}_L$, the set $\mathbf{L}_{\mathcal{C},\mathcal{X},\mathcal{X}_L}$ is finite.

Remark 2. Notice the two distinct meanings of the expressions $[X|Y]$ and $[X|[Y|[]]]$. In both expressions X is a c-term variable denoting the first element of the list, i. .e. a member of the list. In the first expression the variable Y is a list variable denoting the rest of the list, thus representing a list of its own, while in the second expression Y is a c-term variable denoting the second element of the list, i. e. a member of the list. As we do not consider nesting of lists in this paper, these two expressions differ fundamentally in their semantics.

A *term* is either a constant symbol $c \in \mathcal{C}$, a variable symbol $X \in \mathcal{X} \cup \mathcal{X}_L$ or a list $l \in \mathbf{L}_{\mathcal{C},\mathcal{X},\mathcal{X}_L}$. An *atom* A is a predicate symbol P with arity n followed by a sequence of n terms enclosed in parentheses. A *literal* is either an atom A or a negated atom $\neg A$. A literal L is said to be *ground*, if no variable symbol appears in L either as a term or as a member of a list. Let not denote the default negation.

Definition 13 (Extended logic program with lists). An *extended logic program with lists* – or *program* for short – is a finite set of rules of the form

$$A \leftarrow A_1, \ldots, A_n, \mathsf{not}\ B_1, \ldots \mathsf{not}\ B_m.$$

with literals A,A_1, ..., A_n, B_1, ..., B_m.

A.1.2 *Semantics*

Let P be a program and $\mathcal{C}$ be the set of constants appearing in P either as a term or as a member of a list. The grounding P_g of P is determined by replacing all variable symbols appearing in P with the constant symbols in $\mathcal{C}$ and the lists in $\mathbf{L}_{\mathcal{C},\emptyset,\emptyset}$ as follows.

Definition 14 (Grounding). Let P be a program and $\mathcal{C}$ the set of all constants appearing in P. The *grounding* P_g of P is obtained by replacing each rule $r \in P$ with its grounded versions. A *grounded version* r' of a rule $r \in P$ is obtained by

1. substituting each variable X appearing not inside a list in r with a term $t \in \mathcal{C} \cup \mathbf{L}_{\mathcal{C},\emptyset,\emptyset}$,
2. substituting each c-term variable X appearing in a list in r with a term $t \in \mathcal{C}$, and
3. substituting each list variable X appearing in a list in r with a list $l \in \mathbf{L}_{\mathcal{C},\emptyset,\emptyset}$

such that every list appearing in r' is in $\mathbf{L}_{\mathcal{C},\emptyset,\emptyset}$.

The last condition in the above definition ensures that all lists appearing in a grounded version r' of a rule r are valid in the sense, that no list member is a list itself and every element does not appear more that once.

Remark 3. The grounding P_g of a program P satisfies the following properties:

1. P_g is finite.
2. P_g does not contain any variable.

Example 7. Suppose $\mathcal{C} = \{a, b\}$ and consider the rule

$$R(X, [Y|Z]) \leftarrow P(X), Q([Y, a]), R(Z).$$

In a first step the variable X is substituted with a term $t \in \mathcal{C} \cup \mathbf{L}_{\mathcal{C},\emptyset,\emptyset} = \{a, b, [a, b], [b, a], [a], [b], []\}$ yielding among others

$$R(b, [Y|Z]) \leftarrow P(b), Q([Y, a]), R(Z).$$

Then the c-term variable Y is substituted with a term $t \in \mathcal{C}$ yielding $(t = b)$

$$R(b, [b|Z]) \leftarrow P(b), Q([b, a]), R(Z).$$

Notice that Y cannot be substituted by a as this would produce the expression $[a, a]$ which is not a valid list in our sense. Finally, the list variable Z is substituted with a list $l \in \mathbf{L}_{\mathcal{C},\emptyset,\emptyset}$ yielding among others

$$R(b, [b|[a]]) \leftarrow P(b), Q([b, a]), R([a]).$$

Notice that in the final step the list variable Z cannot be substituted by the list $[a, b]$ because then the expression $[b|[a, b]]$ appears in r' which is not a member of $\mathbf{L}_{\mathcal{C},\emptyset,\emptyset}$.

Given the grounding P_g of a program P the semantics of P is defined via answer sets as for general extended logic programs. For that reason, assume that all lists appearing in P_g are in normalized form. This can be achieved by a simple preprocessing step.

Definition 15 (Reduct). Let P_g be the grounding of a program P and let K be a set of grounded literals. The *reduct* P^K of P with K is the logic program without default negation given by

$$\begin{aligned} P^K \quad = \quad & \{A \leftarrow A_1, \ldots, A_n \mid A \leftarrow A_1, \ldots, A_n, \mathsf{not}\ B_1, \ldots \mathsf{not}\ B_m \in P_g, \\ & \qquad K \cap \{B_1, \ldots, B_m\} = \emptyset\} \end{aligned}$$

Definition 16 (Answer set). A set A of grounded literals is an *answer set* of a program P, iff A is the minimal model of P^A.

For the task of computing the answer sets of the grounded program P_g lists can be treated as ordinary constant symbols. Thus standard techniques for computing answers sets for extended logic programs can be used to obtain answer sets for extended logic programs with lists.

BIBLIOGRAPHY

[1] Jose Julio Alferes, Luis Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. LUPS - A language for updating logic programs. *Artificial Intelligence*, 138(1-2):87–116, June 2002.

[2] Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah Seghrouchni, Jorge J. Gomez-Sanz, João Leite, Gregory O'Hare, Alexander Pokahr, and Alessandro Ricci. A survey of programming languages and platforms for multiagent systems. *Informatica*, 30:33–44, 2006.

[3] Rafael H. Bordini, Jomi F. Hübner, and Renata Vieira. Jason and the golden fleece of agent-oriented programming. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming Languages, Platforms and Applications*, chapter 1, pages 3–37. Kluwer Academic Publishers, 2005.

[4] Michael Brenner. Continual collaborative planning for mixed-initiative action and interaction. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, pages 1371–1374, 2008.

[5] Hans Chalupsky, Tim Finin, Rich Fritzson, Don McKay, Stu Shapiro, and Gio Weiderhold. An overview of KQML. Technical report, KQML Adv. Group, 1992.

[6] Michael A. Covington, Donald Nute, and Andre Vellino. *Prolog Programming in Depth*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[7] T. Eiter, W. Faber, C. Koch, N. Leone, and G. Pfeifer. DLV - a system for declarative problem solving. In C. Baral and M. Truszczynski, editors, *Proc. of the 8th Int. Workshop on Non-Monotonic Reasoning*, 2000.

[8] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. Planning under incomplete knowledge. In *Proc. of the First Int. Conf. on Computational Logic*, number 1861 in LNCS/LNAI, pages 807–821, 2000.

[9] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. Reasoning about evolving nonmonotonic knowledge bases. *ACM Trans. Comput. Logic*, 6(2):389–440, 2005.

[10] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intel.*, 2(3-4):189–208, 1971.

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

[12] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[13] G. Kern-Isberner. *Conditionals in nonmonotonic reasoning and belief revision*. Number 2087 in Lecture Notes in Computer Science. Springer, 2001.

[14] Patrick Krümpelmann and Gabriele Kern-Isberner. Propagating credibility in answer set programs. In *Proceedings of the 22nd Workshop on (Constraint) Logic Programming (WLP08)*, Dresden, Germany, October 2008.

[15] Patrick Krümpelmann, Matthias Thimm, Manuela Ritterskamp, and Gabriele Kern-Isberner. Belief Operations for Motivated BDI Agents. In Lin Padgham, David C. Parkes, Joerg P. Müller, and Simon Parsons, editors, *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 421–428, Estoril, Portugal, May 2008.

[16] Yves Lesperance, Giuseppe De Giacomo, and Atalay Ozgovde. A model of contingent planning for agent programming languages. In *Proceedings of AAMAS'08*, pages 477–484, 2008.

[17] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intel.*, 138(1-2):39–54, June 2002.

[18] Fangzhen Lin and Yisong Wang. Answer set programming with functions. In Gerhard Brewka and Jerome Lang, editors, *Proc. of the Eleventh Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 454–464, 2008.

[19] M. Luck and M. d'Inverno. Motivated behaviour for goal adoption. In Zhang and Lukose, editors, *Proc. of the 4th Australian Workshop on Distributed Artificial Intelligence*, pages 58–73. Springer, 1998.

[20] T. J. Norman and D. Long. Goal creation in motivated agents. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages*, pages 277–290. Springer, 1995.

[21] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN 19, Department of Computer Science, Aarhus University, Aarhus, Denmark, 1981.

[22] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A short overview. In *Main Conference Net.ObjectDays 2004, AgentExpo*, 2004.

[23] A. S. Rao and M. P. Georgeff. BDI-Agents: From theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[24] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, San Mateo, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[25] M. Schut, M. Wooldridge, and S. Parsons. The theory and practice of intention reconsideration. *Journal of Experimental and Theoretical Artificial Intelligence*, 16(4):261–293, December 2004.

[26] M. P. Singh. Know-how. In Anand S. Rao and M. J. Wooldridge, editors, *Foundations of Rational Agency, Applied Logic Series*, pages 105–132. Kluwer, 1999.

[27] M. P. Singh, A. S. Rao, and M. P. Georgeff. Formal methods in DAI: Logic-based representation and reasoning. In G. Weiss, editor, *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, pages 331–376. The MIT Press, Cambridge, Massachusetts, 1999.

[28] John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI*, pages 721–726. Academic Press, 2003.

[29] M. Thielscher. Flux: A logic programming method for reasoning agents. *Theory And Practice of Logic Programming*, 2004.

[30] Wiebe van der Hoek and Michael Wooldridge. Towards a logic of rational agency. *Logic Journal of IGPL*, 11(2):135–159, 2003.

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

[32] Shung-Bin Yan, Zu-Nien Lin, Hsun-Jen Hsu, and Feng-Jian Wang. Intention scheduling for BDI agent systems. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, volume 1, pages 133–140, 2005.