A Logic Programming Framework for Reasoning about Know-How

Patrick Krümpelmann and **Matthias Thimm**
Information Engineering Group, Technische Universität Dortmund, Germany

Abstract

The BDI model is a well accepted architecture for the representation of rational agents, both at theoretical and practical levels. The beliefs component in this model is focused on the representation of beliefs about the world and other agents and widely independent of an agent's intentions. We argue that also the logical representation of an agent's intentions and its know-how, which captures the beliefs about actions and procedures, has to be taken into account when modeling rational agents. With a declarative rather than procedural representation of know-how we obtain the possibility to reason with these procedural beliefs in the same way as with any other logical beliefs. Using the notion of know-how as introduced by Singh we formalize a usable and concrete agent architecture that benefits from this representation of procedural beliefs in multiple ways. To model both, the agent's logical beliefs as well as its know-how, we make use of extended logic programs under the answer set semantics that are capable of handling uncertain information. This way we are lifting the limitations imposed by the use of different forms of representation. We define a general algorithm that uses this representation for means-end reasoning. Furthermore, we present diverse ways of reasoning under uncertainty about know-how that are enabled by our framework and show relations to existing approaches.

Introduction

Planning and agency are two closely related fields in the research of multiagent systems and artificial intelligence in general. While planning (Ghallab, Nau, and Travers 2004) deals with solving problems by constructing sequences of atomic actions that lead to a solution, the theory on agents (Weiss 1999) is concerned with how to exhibit rational behavior in complex interaction scenarios. From the perspective of knowledge representation and reasoning under uncertainty a proper mutual integration is desirable but not yet available. The most prominent example paradigm for representing rational agents is the BDI model (Weiss 1999) which is widely known and used. This model divides the mental state of an agent into *beliefs*, *desires*, and *intentions*. Informally speaking, *beliefs* comprise the agent's beliefs about the world and the current situation, *desires* represent what it wishes to achieve and hence represent possible goals, whereas *intentions* model the agent's immediate (sub-) goals. Intentions thus focus on the next actions the agent should undertake in order to achieve the current goal. In modern formalizations of the BDI model or other agent architectures for planning and reasoning (Bordini, Hübner, and Vieira 2005; Bordini et al. 2006; Lifschitz 2002; Thielscher 2004) planning and belief components are mostly kept separate. Although beliefs do (of course) influence intention deliberation and goal generation, the other way round cannot be formalized in a natural way in these systems. In particular, in most systems it is not possible to reason about the current intentions or capabilities of the agent on the same level where the classic logical reasoning of the agent takes place.

Singh introduced an abstract notion of *know-how* in (Singh 1999) and 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 properly. But as know-how is a form of belief, an agent should be able to reason about these beliefs and to revise them. Singh (Singh 1999) 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*". 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. We argue that in many practical scenarios with uncertain information it is necessary and advisable for the agent to reason about its planning capabilities on the *object-level*, while in modern agent architectures (Bordini et al. 2006) this can at most be done on a *meta-level*.

Example 1. Imagine a cleaner robot that pursuits two goals, namely cleaning all rooms in a certain area and maintaining 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. Likewise, 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.

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 (Lesperance, Giacomo, and Ozgovde 2008; Brenner 2008). Current research includes some ideas of how to com-

bine state-of-the-art knowledge representation and reasoning techniques with problem solving and planning, e. g. using logic programming (Lifschitz 2002). Here, we take a look at the planning capabilities of an agent from the perspective of knowledge representation. By modeling know-how in a declarative rather than procedural manner we enable the agent to reason about and revise these beliefs in the same way as any other of its logical beliefs. To this end we facilitate extended logic programs under the answer set semantics (Gelfond and Leone 2002). Extended logic programs proved to be very suitable for knowledge representation and reasoning and are widely used in a variety of applications which qualifies them as a powerful basis for our approach.

We formalize the idea of know-how in logic programming and develop a general algorithm, which enables an agent to make use of complex know-how structures. Moreover, we show how agents can reason about know-how facilitating the possibilities opened by our framework. Thus, we present a full-fledged framework for advanced representation and handling of as well as reasoning about structural knowledge.

This paper is organized as follows. In Section 2 we give some background on extended logic programs under the answer set semantics, know-how and a short overview on the overall agent architecture. We continue in Section 3 with the description of the main framework which comprises the representation of know-how in extended logic programs and a general algorithm for means-end reasoning. Section 4 gives some hints on the versatility of the framework with respect to reasoning. In particular, we show how to determine sound and reliable know-how and how to apply belief revision on know-how. In Section 5 we conclude and discuss our approach in the context of related work.

Background

For our framework we make use of extended logic programs under the answer set semantics (Gelfond and Leone 2002) which distinguish between classical negation "$\neg$" and default negation "not " and which are capable of dealing with incomplete information in open environments. Let At be a set of atoms and $\mathcal{L}$ the corresponding set of literals, i. e., for every $a \in At$ it is $a \in \mathcal{L}$ and $\neg a \in \mathcal{L}$. Whenever necessary we use first-order predicates and terms, i. e. constants (starting with a lowercase letter) and variables (starting with an uppercase letter), to build atoms and literals in the usual way. An *extended logic program* P is a finite set of rules of the form

$$r : h \leftarrow a_1, \ldots, a_n, \mathsf{not}\ b_1, \ldots, \mathsf{not}\ b_m.$$

where $h, a_1, \ldots, a_n, b_1, \ldots, b_m \in \mathcal{L}$. We denote by $\mathcal{H}(r)$ the head h and by $\mathcal{B}(r)$ the body $\{a_1, \ldots, a_n, \mathsf{not}\ b_1, \ldots, \mathsf{not}\ b_m\}$ of a rule r. If the body of a rule r is empty ($\mathcal{B}(r) = \emptyset$), then r is called a *fact*, abbreviated h instead of $h \leftarrow$. Given a set $X \subseteq \mathcal{L}$ of literals, then r is *applicable* in X, iff $a_1, \ldots, a_n \in X$ and $b_1, \ldots, b_m \notin X$. The rule r is *satisfied* by X, if $\mathcal{H}(r) \in X$ or if r is not applicable in X. X is a model of an extended logic program p iff all rules of P are satisfied by X. The set $X \subseteq \mathcal{L}$ is *consistent*, iff for every $a \in At$ it is not the case that $a \in X$ and $\neg a \in X$.

An answer set is a minimal consistent set of literals that satisfies all rules. Let P be an extended logic program and $X \subseteq \mathcal{L}$ a set of literals. The *X-reduct* of P, denoted P^X, is the union of all rules $h \leftarrow a_1, \ldots, a_n.$ such that $h \leftarrow a_1, \ldots, a_n, \mathsf{not}\ b_1, \ldots, \mathsf{not}\ b_m. \in P$ and $X \cap \{b_1, \ldots, b_m\} = \emptyset$. For any extended logic program P and a set X of literals, the X-reduct of P is a logic program P' without default negation and therefore has a minimal model. If P' is inconsistent, then its unique model is defined to be $\mathcal{L}$. Let P be an extended logic program. A consistent set of literals $S \subseteq \mathcal{L}$ is an *answer set* of P, iff S is a minimal model of P^S.

We extend the syntax of extended logic programs with lists as in Prolog. A *list* is a sequence of terms enclosed by squared brackets, e. g. $[a, b, c]$, that can appear as a parameter for predicates like any other term. We use the notion $[H|B]$ 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 used here) it can be shown, that extended logic programs with lists can be rewritten in extended logic programs without lists, such that the answer sets of the rewritten program can be appropriately rewritten to answer sets with lists (Lin and Wang 2008; Thimm and Krümpelmann 2009).

In our framework we use extended logic programs to represent beliefs, intentions and the know-how of an agent. Although our overall agent architecture is based on a BDI model we neglect the *desires* of an agent here to simplify presentation. Therefore, we consider only a single top-level goal that is initially the only element of the agent's intentions and reduce the complexity of the deliberation process, cf. Figure 1. Whenever needed we refer to the agent's intentions as a stack where the topmost intention is the intention currently pursued by the agent and subgoals are stacked on top of their super-goals. Some intentions, called *atomic intentions*, can be directly fulfilled by executing a corresponding action in the environment.

An agent is situated in an infinite loop of perception, deliberation and action, as Figure 1 illustrates. At the beginning of the loop the agent integrates newly perceived information into its beliefs and correspondingly updates its intentions, cf. (Krümpelmann et al. 2008; Krümpelmann and Kern-Isberner 2008). After having integrated newly perceived information the agent deliberates about its current status using information about what it knows (beliefs), what it wants to achieve (intentions), and its capabilities (know-how). Eventually it executes an action in the environment. A comprehensive integration of the techniques described here into a complete BDI model can be found in (Thimm and Krümpelmann 2009).

We presuppose that the agent's knowledge about procedures, i. e. its know-how, can be structured into atomic statements σ of the form

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

with abstract labels a (the goal of the statement), $s_1, \ldots, s_n$

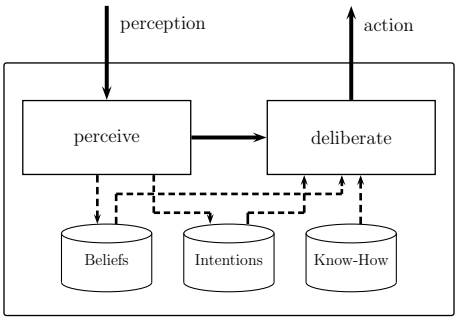

Figure 1: A simplified representation of the agent model. (The control flow is depicted with solid lines and the data flow with dashed lines.)

(the sub-goals of the statement), and $c_1, \ldots, c_m$ (the conditions of the statement). The informal meaning of such a statement σ is as follows: In order to achieve the intention a the agent can try to achieve the intentions $s_1, \ldots, s_m$, if the conditions $c_1, \ldots, c_m$ are fulfilled with respect to the agent's beliefs. Roughly, this kind of representation follows the usual form of procedures as in, e. g., STRIPS (Ghallab, Nau, and Travers 2004).

Example 2. We continue Example 1. Suppose the robot has the top-level goal (desire) to clean all rooms in its area which is represented by $cleaned\_all$. 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 statement:

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

With the use of these statements the agent can break down complex intentions into simpler ones which leads to the execution of an action. We give a more detailed description of this procedure and the actual representation of know-how and intentions using logic programs in the following sections.

Know-How in Logic Programming

In the following we develop a framework for the representation of beliefs, intentions, and know-how in the sense described above using extended logic programming. Before stating the definitions of know-how, we need some more notation. Let $\mathcal{C}$ be a set of constant symbols with $\mathcal{C} = \mathcal{C}_{kh} \dot\cup \mathcal{C}_s$. For what is coming constants from $\mathcal{C}_{kh}$ are used to identify know-how statements using reification and constants from $\mathcal{C}_s$ are used to represent logical statements, e. g. conditions or intentions. KB denotes the (logical) belief base of an agent which is an extended logic program. As outlined in the previous section, a know-how statement is comprised of an identifier σ, a goal a, a sequence of subgoals $s_1, \ldots, s_n$, and a set of conditions $c_1, \ldots, c_m$.

Definition 1 (Know-How Statement)**.** A *know-how statement* is a logic program of the form:

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

with constant symbols $\sigma \in \mathcal{C}_{kh}$, $a, s_1, \ldots, s_n, c_1, \ldots, c_m \in \mathcal{C}_s$ and $m, n \in \mathbb{N}$.

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

Definition 2 (Know-How Base)**.** Let $\{K_1, \ldots, K_n\}$ be a set of know-how statements. A *know-how base* Σ is an extended logic program defined as

$$\Sigma = \bigcup_{i=1}^{k} K_i \cup \{is\_atomic(int). \mid int \in \mathcal{C}_s, \; int \text{ is an atomic intention}\}.$$

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 with constants using reification, while the logical beliefs of the agent are modeled by means of literals and rules. For example, the representation of the statement in Example 2 as a know-how statement contains besides others the logical fact $khCondition(\sigma, battery\_full)$, while the logical beliefs of the agent may contain the fact $BatteryFull$. In order to use the logical beliefs to check for conditions, we assume that for each constant $c \in \mathcal{C}_s$ there is also the rule

$$holds(c) \leftarrow L.$$

in the logical beliefs where L is the corresponding literal, that is identified with the constant c. For example, for the constant $battery\_full$ we add the rule

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

to the agents beliefs. We only introduce these rules for literals of arity zero as we only allow conditions to be constants. For literals with arity greater zero these mechanism can be extended in order to allow conditions as well as goals and sub-goals to be parametrized. But to keep our presentation simple 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.

We continue by constructing an algorithm in logic programming which works on the representation developed above. This algorithm takes an agent's mental state, returns the next atomic action as output and, as a side effect, modifies the agent's intentions according to its deliberations. The agent's mental state is represented as a collection of extended logic programs which comprises the agent's (logical) beliefs KB, the agent's know-how base Σ, and the logic programs defined below. We define a logic program I that represents the intentions of the agent and keeps track of the current deliberation step. The program contains a set of facts for each step of its deliberation as follows:

- $istack(is)$: is captures the intention stack of the agent as a list, the first intention being the currently pursued intention.

- $khstate(ks)$: ks is a list that represents the current stack of know-how statements corresponding to the intention stack.
- $act(ai)$: if the agent determined an atomic intention, ai, to be executed, in the previous step of plan deliberation, this fact is added to I.
- $khFailed(kf)$: kf is a list indicating, that the first know-how statement failed in the context of the rest of kf due to unsatisfied conditions.
- $khPerformed(kh)$: this fact is added to I if all subgoals of the specified know-how statement were fulfilled in the previous step of plan deliberation.
- $state(s)$: describes the current state of plan deliberation by summarizing the operation of the last step. Accordingly, s 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 $khstate$)
  - *noop* (no operation has been performed)

The facts $istack$, $khstate$ and $khFailed$ are the main components of I and describe the agent's current deliberation process. Initially the logic program I of an agent has the structure

$$istack([initial\_intention]).$$
$$khstate([]).$$
$$state(\textit{intentionAdded}).$$

with a given initial intention $initial\_intention$. Given a know-how base Σ and the logic program I, the logic program `NextAction` defined below uses these programs by generating a new state for I as part of the answer sets. Using these programs the algorithm depicted in Figure 2 computes the next action for the agent by repeatedly computing the intersection[1] of all answer sets of the union of I, the know-how base Σ, the (logical) belief base KB and the rules in `NextAction`.

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

Figure 2: The algorithm for plan deliberation

Definition 3. Let `NextAction` be the logic program that contains rules (r_1) to (r_{15}):

[1] We base our algorithm on a skeptical inference process. One can also imagine an algorithm based on credulous inference, i. e. taking a specific answer set as result.

Rules for State *intentionAdded*:

(r_1) $new\_act(A) \leftarrow istack([A|H]), is\_atomic(A), state(\textit{intentionAdded}).$
(r_2) $new\_khstate([KH|K]) \leftarrow khstate(K), khStatement(KH, I), istack([I|\_]),$ not $khConditionsFail(KH), khFailed([KH|K]), state(\textit{intentionAdded}),$ not $new\_act(\_).$
(r_3) $new\_khFailed([KH|K]) \leftarrow state(\textit{intentionAdded}),$ not $new\_act(\_),$ not $new\_khState(\_), khState([KH|K]).$
(r_4) $toParent \leftarrow new\_khFailed(\_).$

Rules for State *actionPerformed*:

(r_5) $new\_istack([C|B]) \leftarrow khstate([KH|\_]), state(\textit{actionPerformed}), istack([A|B]), khSubgoal(KH, I, A), J = I + 1, khSubgoal(KH, J, C).$
(r_6) $new\_khPerformed(KH) \leftarrow khstate([KH|\_]), state(\textit{actionPerformed}), istack([A|\_]), khSubgoal(KH, I, A), J = I + 1,$ not $khSubgoal(KH, J, \_).$
(r_7) $toParent \leftarrow new\_khPerformed(\_).$

Rules for State *khAdded*:

(r_8) $new\_istack([I|B]) \leftarrow istack(B), khSubgoal(KH, 1, I), state(\textit{khAdded}), khstate([KH|K]).$

State transition rules:

(r_9) $new\_state(\textit{actionPerformed}) \leftarrow new\_act(A).$
(r_{10}) $new\_state(\textit{intentionAdded}) \leftarrow new\_istack(\_),$ not $new\_khPerformed(\_).$
(r_{11}) $new\_state(\textit{khAdded}) \leftarrow new\_khState(\_),$ not $new\_khPerformed(\_).$
(r_{12}) $new\_state(\textit{intentionAdded}) \leftarrow new\_khFailed(\_),$ not $istack([]).$
(r_{13}) $new\_state(\textit{noop}) \leftarrow new\_istack([]).$

Auxiliary rules:

(r_{14}) $new\_istack(B) \leftarrow toParent, istack([\_|B]).$
(r_{15}) $new\_khState(K) \leftarrow toParent, khState([\_|K]).$
(r_{16}) $khConditionsFail(KH) \leftarrow khCondition(KH, X),$ not $holds(X).$

Due to the single instantiation of $state(\cdot)$ in the initial program I and the structure of the rules for literals of the form $new\_action(\cdot)$, $new\_istack(\cdot)$, $new\_khstate$, etc. it holds: if one of these literals appears in an answer set, then it appears in all answer sets (given that the logical beliefs in KB do not influence this derivation in an undesired manner). In the algorithm depicted in Figure 2 these literals are extracted from the intersection, their "$new\_$" prefixes are stripped off and they are set as the new program I.

The following propositions show that the program `NextAction`, the interaction of the rules, and the iterative computation of answer sets behave well. Let $\Omega =$

(KB, Σ, I) denote the mental state of an agent.

Proposition 1. *It holds:*

1. *For every mental state Ω of an agent, there is not more than one new mental state Ω' induced by the answer sets of $\texttt{NextAction} \cup \Omega$.*
2. *For every mental state Ω of an agent—except a mental state with $state(noop) \in I$—there is a valid mental state Ω' induced by the answer sets of $\texttt{NextAction} \cup \Omega$.*

Proof sketch. Given some mental state with $state(intentionAdded) \in I$ one of the rules (r_1)-(r_3) is applicable. These are mutual exclusive as for each pair of rules the head of one rule is contained in the negative body of the other. If rule (r_1) is applicable then rule (r_9) is also automatically applicable and the applicability of (r_2) leads to the applicability of (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 applicability 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. Hence, (r_4) is applicable which resembles a $toParent$ operation by means of the applicability of rules (r_{14}) and (r_{15}). At this point (r_{12}) will be applicable in the same iteration and marks the successor mental state by deriving $state(intentionAdded)$. 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. The know-how statement which 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 and terminates if (r_2) is finally applicable or if the intention stack is empty and (r_{13}) is satisfied and the state *noop* is reached. If $state(\textit{actionPerformed}) \in I$ one of the rules $(r_5) - (r_6)$ can be applicable which are mutual exclusive again. If (r_5) is applicable the subsequent subgoal is placed on the list and by means of (r_9) the state changes to *actionPerformed*. In the case that (r_6) is applicable, (r_7) will be applicable as well and a $toParent$ operation will be executed by means of rules (r_{14}) and (r_{15}). The state transition rules are blocked at this point as the state stays the same and the parent intention will be considered in the next iteration. If $state(khAdded) \in I$ then (r_8) will be applicable and adds the first subgoal of the added know-how statement to the intention stack which again leads to the applicability of (r_{10}). The other subgoals are captured by the $khSubgoal(\cdot, \cdot, \cdot)$ predicate and that they will be added by means of the rules (r_5). □

Proposition 2. *Every iterated determination of a mental states terminates in a mental state with $state(noop) \in I$.*

A more comprehensive description on the algorithm and the complete proofs can be found in (Thimm and Krümpelmann 2009). This representation of know-how and the algorithm for means-end reasoning constitute the basis of a versatile framework for know-how that we illustrate in the following.

Reasoning about Know-How

One of the main motivations for the explicit representation of know-how and intentions in a declarative manner is that this enables the agent to reason about these components. Moreover, the agents awareness of its procedural knowledge can deeply influence the behaviour of its means-end reasoning. The structure of know-how reveals information about the involved conditions and subgoals of the achievement of goals and this information enables the agent to reason about the feasibility and effort as well of the reliability of the achievement of goals, as we will demonstrate in this section.

Sound Know-How

Under most circumstances it is reasonable that an agent can determine if it has the means to achieve a given intention before it starts acting towards its fulfillment. One simple realization of this idea is captured by the following definition.

Definition 4 (Sound Know-How)**.** Let Σ be a know-how base. An intention $i \in \mathcal{C}_s$ is *achievable* in Σ if

- i is an atomic intention or
- there is at least one $\sigma \in \mathcal{C}_{kh}$ with $khStatement(\sigma, i) \in \Sigma$ and every s with $khSubgoal(\sigma, \cdot, s) \in \Sigma$ is achievable.

Σ is called *sound* if every intention $i \in \mathcal{C}_s$ is achievable.

Given an know-how base Σ the rules stated in Figure 3 determine whether an intention is achievable. If these rules are added to the agent's beliefs the agent is aware of its capabilities at any time and can use this information to avoid unnecessary actions.

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

Figure 3: Determination of sound know-how

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. We will discuss a notion called *reliable know-how* in the next subsection which will take these conditions into consideration.

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 (Singh 1999). He also notes that alternative, less strict, versions of reliability might be formulated. A similar notion is the one of *secure planning* (Eiter et al. 2000), 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'

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

Figure 4: Determination of reliable know-how

actions. 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 know-how. For simplicity, 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 C, i. e. a subset of $\mathcal{C}_s$.

Definition 5 (Reliability)**.** Let Σ be a know-how base, $i \in \mathcal{C}_s$ an intention and $C \subseteq \mathcal{C}_s$. The intention i is *reliably achievable in* C if i is an atomic intention or if there is at least one $\sigma \in \mathcal{C}_{kh}$ with $khStatement(\sigma, i) \in \Sigma$ and σ is reliable in C. A $\sigma \in \mathcal{C}_{kh}$ with $khSubgoal(\sigma, 1, s_1) \ \ldots \ khSubgoal(\sigma, n, s_n) \in \Sigma$ is *reliable in context* C iff 1.) each s_i $(1 \leq i \leq n)$ of σ is reliably achievable in $C \cup \{s_1, \ldots, s_{i-1}\}$ and 2.) the conditions of σ are fulfilled in C.

Note that the intentions of previous sub-goals are added to the context of a sub-goal as well. This is being done as previous actions may be requisite for application of later intentions. Our notion of reliable know-how is implemented through the addition of the rules depicted in Figure 4 to the agent's beliefs. These rules enable the agent to always have knowledge about which of its know-how is reliable and which is not. 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 intentions. In the example of the cleaner robot this means that it keeps on cleaning as long as it reliably knows how to get to the charging station.

Revision and Update of Know-How

Communicating agents in a dynamic environment need the capability to adopt their beliefs when they acquaint new information. Usually these changes of beliefs are limited to logical beliefs. In terms of belief dynamics, the logical representation of know-how comes with the benefit that know-how can be subject to dynamic changes by use of the same operators as for logical beliefs. The agent can revise its know-how when it receives new know-how statements and thus can incorporate them into the existing know-how base while maintaining consistency. Due to the representation on the same level with other beliefs, the dynamics of logical beliefs and of know-how are processed at the same time using the same mechanism and thus respecting interactions of both. Moreover, updates of subgoals or conditions of know-how statements are automatically performed if new subgoals or conditions are added for a given know-how statement. This is due to conflicts which can be a direct one which is handled by update mechanisms directly. Also, indirect conflicts can be used for revision as they can easily be made obvious by use of integrity constraints of the form

$$
\begin{array}{lll}
 & & khSubgoal(KH,I,Int1), \\
\leftarrow & & khSubgoal(KH,I,Int2), \\
 & & Int1 \neq Int2.
\end{array}
$$

as these impose conflicts in the sense of answer set semantics which need to be solved. Some modifications of update mechanism for logic programs like (Krümpelmann and Kern-Isberner 2008) towards the consideration of conflicts induced by constraints have to be made for this matter which we do not explicate here due to space restrictions. We will, however, exemplify some of the possibilities in the following example.

Example 3. Building on Example 2 we can consider different types of new information the cleaner robot might acquire. For instance, assume that the robot's assignment has been changed so that it now has to clean the kitchen instead of the lounge. This results in the addition of a new fact of the form $khSubgoal(\sigma, 2, cleaned\_kitchen)$ into the (procedural) beliefs of the robot. This fact will lead to a conflict induced by the constraint defined above. Assuming a change operator for logic programming based on causal rejection (Krümpelmann and Kern-Isberner 2008) this will lead to the rejection of the older piece of information in conflict, in this case the fact $khSubgoal(\sigma, 2, cleaned\_lounge)$.

The agent might also receive the information that it has to consider the status of its bag in form of the fact $khCondition(\sigma, bag\_empty)$. This resembles a pure expansion, as the addition of this fact does not lead to any conflict. In case the agent realizes that there is enough light for its solar panels it might want to stop caring about its battery status and the fact $\neg khCondition(\sigma, battery\_status)$ is added to its beliefs. This induces a direct conflict with the fact $khCondition(\sigma, battery\_status)$ which is the less recent information and will thus be rejected. The strict negated predicate $\neg khCondition(\sigma,\ battery\_status)$ is out of the scope of the know-how statement σ and therefore this operation resembles a contraction in terms of know-how.

More complex change operations to know-how statements can be implemented by simply adding some rules which will take care of the effects of initiated changes, like keeping the numbering of know-how subgoals in a consistent state. Updates of know-that will instantaneously lead to changes to the reliability of know-how which again will influence other components of the agent as has been laid out earlier. Another aspect in this manner is the communication and transfer of know-how (Krümpelmann et al. 2008).

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. Know-how, as represented in our framework, can easily be subject to dynamic changes just like any other logical beliefs of a rational agent in a dynamic multiagent environment.

Discussion

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 implementing this abstract notion. In particular, we use extended logic programs for the representation of the agents beliefs extended by declarative knowledge of know-how. Through this, 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.

Our approach of breaking down meta-level reasoning on intention deliberation to the object level is similar to the approach undertaken in (Meneguzzi and Luck 2007). There, the abstract notion of *motivation* is used to connect declarative properties of an agent, i e. its motivation, to its component for meta-level reasoning. While this approach allows a powerful control mechanism for the meta-level component it does not enable the agent to reason about this component on the object-level itself. There is also a wide range of agent architectures that built on the BDI model (Bordini et al. 2006). All these architectures do not inherently support the notion of know-how nor reasoning about intentions and plan deliberation in the way we present here. The system Jason (Bordini, Hübner, and Vieira 2005), also, allows a simple treatment of revision of plans in the sense, that plan fragments may be added or deleted from the agent program. We believe that our proposal can by adapted for several agent architectures in order to enrich them with a notion of know-how and reasoning capabilities about this. Extended logic programming has been extensively used as a language for plan deliberation. For example, in (Lifschitz 2002) Lifschitz uses extended logic programming in order to to provide solutions for the blocks world problem. In contrast to (Lifschitz 2002), 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 $\mathcal{K}$ (Eiter et al. 2000) also makes use of extended logic programs as representation technique and features a notion of *reliability* called *secure plans* as well. But like in other agent systems, $\mathcal{K}$ does not allow the treatment of structural knowledge about planning capabilities as ordinary logical beliefs.

We see our proposal as a first step to the full support of the notion of know-how (Singh 1999) 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.

References

Bordini, R. H.; Braubach, L.; Dastani, M.; Seghrouchni, A. E. F.; Gomez-Sanz, J. J.; Leite, J.; O'Hare, G.; Pokahr, A.; and Ricci, A. 2006. A survey of programming languages and platforms for multiagent systems. *Informatica* 30.

Bordini, R. H.; Hübner, J. F.; and Vieira, R. 2005. Jason and the golden fleece of agent-oriented programming. In Bordini, R. H.; Dastani, M.; Dix, J.; and Seghrouchni, A. E. F., eds., *Multi-Agent Programming Languages, Platforms and Applications*. Kluwer. chapter 1, 3–37.

Brenner, M. 2008. Continual collaborative planning for mixed-initiative action and interaction. In *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems*.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2000. Planning under incomplete knowledge. In *Proc. of the First Int. Conf. on Computational Logic*, 807–821.

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

Ghallab, M.; Nau, D.; and Travers, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Krümpelmann, P., and Kern-Isberner, G. 2008. Propagating credibility in answer set programs. In *Proc. of the 22nd Workshop on (Constraint) Logic Programming (WLP08)*.

Krümpelmann, P.; Thimm, M.; Ritterskamp, M.; and Kern-Isberner, G. 2008. Belief operations for motivated BDI agents. In *Proc. of the 7th Int. Conf. on Autonomous Agents and Multiagent Systems*, 421–428.

Lesperance, Y.; Giacomo, G. D.; and Ozgovde, A. 2008. A model of contingent planning for agent programming languages. In *Proceedings of AAMAS'08*, 477–484.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artif. Intel.* 138(1-2):39–54.

Lin, F., and Wang, Y. 2008. Answer set programming with functions. In *Proc. of the Eleventh Int. Conf. on Principles of Knowledge Representation and Reasoning*, 454–464.

Meneguzzi, F. R., and Luck, M. 2007. Motivations as an abstraction of meta-level reasoning. In *Proc. of the Fifth Int. Central and Eastern European Conference on Multi-Agent Systems*, 204–214.

Singh, M. P. 1999. Know-how. In Rao, A. S., and Wooldridge, M. J., eds., *Foundations of Rational Agency, Applied Logic Series*. Kluwer. 105–132.

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

Thimm, M., and Krümpelmann, P. 2009. Know-how for Motivated BDI Agents (Extended version). Technical Report 822, TU Dortmund, Dpt. of Computer Science.

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