Towards Epistemic Planning Agents

Manfred Eppe[1] and Frank Dylla[2]
[1]*Department of Computer Science and Mathematics, University of Bremen, Bremen, Germany*
[2]*Cognitive Systems, SFB/TR8 Spatial Cognition, University of Bremen, Bremen, Germany*

Keywords: Planning, Epistemic Reasoning, Action Theory, Event Calculus.

Abstract: We propose an approach for single-agent epistemic planning in domains with incomplete knowledge. We argue that on the one hand the integration of epistemic reasoning into planning is useful because it makes the use of sensors more flexible. On the other hand, defining an epistemic problem description is an error prone task as the epistemic effects of actions are more complex than their usual physical effects. We apply the axioms of the Discrete Event Calculus Knowledge Theory (DECKT) as rules to compile simple non-epistemic planning problem descriptions into complex epistemic descriptions. We show how the resulting planning problems are solved by our implemented prototype which is based on Answer Set Programming (ASP).

1 INTRODUCTION

Many approaches to AI planning rely on the strong and unrealistic assumption that complete knowledge about the world is available. These planning systems which consider incomplete knowledge usually don't consider conditional action effects and information acquired by sensing is always used directly. We point out that if epistemic reasoning (ER) (Blackburn et al., 2001) is integrated in planning, then sensing provides extra information. Our first hypothesis is: *A planning system which accounts for both epistemic reasoning and actions with conditional effects can be used to compensate for missing or broken sensors and to work around expensive sensing actions using cheaper sensing actions.* For example, if one wants to find out whether a liquid is poisonous it is costly to learn about its poisonousness by drinking the liquid. Instead, conditional effects of actions can be exploited to achieve *indirect sensing*. Assume a robot with the (sub-)goal to know whether a door is open or not. If equipped with an appropriate sensor it will sense the door state and the goal is achieved. If it does not feature this specific door sensor or if the sensor is broken it can still acquire the information indirectly, e.g. through a location sensor it might be equipped with: It tries to drive through the door, senses its location and infers the door's open-state via the location. If it is behind the door then the door is open and if it is still in front of the door, then the door is closed. In the following, we use the term *epistemic planning* (EP) when we speak about a single agent which is aware of having incomplete knowledge about the world, which can sense and which integrates epistemic reasoning in the planning process. A problem in EP is that deriving epistemic representations of planning problems can be cumbersome when done manually (see Section 4.) Our second hypothesis is: *The automated translation of non-epistemic planning problem domains into epistemic ones simplifies the formal description of planning problems.* We also argue that automated translation also guarantees soundness wrt. epistemic theory, and we make the claim that currently there exists no planning system which automatically generates epistemic planning domain descriptions and which considers actions with conditional effects. Our main result is the Planning Language for Incomplete Knowledge (PLIK) and a tool which performs the automated translation of PLIK planning problem descriptions into an epistemic dialect of the Event Calculus (EC) (Kowalski, 1986) using the Discrete Event Calculus Knowledge Theory (DECKT) (Patkos and Plexousakis, 2009) (Figure 1). The epistemic problem description is input of the f2lp-tool by Lee and Palla (2012) which translates the problem description into an answer set programming (ASP) problem. We interpret the solutions of the ASP

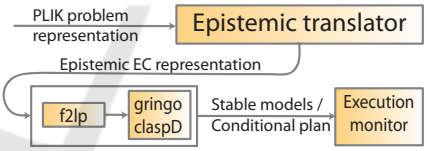

Figure 1: The toolchain of our planning system.

Eppe M. and Dylla F..
Towards Epistemic Planning Agents.
DOI: 10.5220/0004260603110317
In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence* (ICAART-2013), pages 311-317
ISBN: 978-989-8565-39-6

problem as conditional plans. For details which we omit in this paper due to space restrictions we refer to (Eppe and Dylla, 2012).

2 RELATED WORK

The Model Based Planning (MBP) system by Bertoli et al. (2001) is closely related to our work, as it implicitly accounts for the knowledge-level effects of actions. That means, one can specify conditional effects of actions in its input language NPDDL (Bertoli et al., 2002) and the planner handles their epistemic effects. However, the authors do not prove that their underlying action theory is epistemically sound. The PKS planner by Petrick and Bacchus (2004) is the only system we know which explicitly regards for knowledge-level effects of actions. Nevertheless, knowledge-level effects must be handled manually and demand a complex problem specification.

Planners demand for action theories which describe properties of the world and how they change. Prominent examples are the Situation Calculus (SC) (Reiter, 2001), Event Calculus (EC) (Kowalski, 1986) and Fluent Calculus (FC) (Thielscher, 1998). Action theories involving epistemic reasoning are introduced by (Moore, 1985), who describes the possible-worlds semantics of knowledge using Kripke structures and an epistemic K-fluent. (Scherl and Levesque, 2003) continue this work and solve the frame problem for epistemic SC. IndiGolog (De Giacomo and Levesque, 1998) and FLUX (Thielscher, 2005) are high-level programming languages based on SC and FC. In both languages it is possible to express epistemic effects of actions but these effects have to be implemented manually and their epistemic accuracy is not guaranteed.

Our work is based on the Event Calculus (EC) by (Kowalski, 1986) and the Discrete Event Calculus Knowledge Theory (DECKT) by (Patkos and Plexousakis, 2009). The theory uses the predicate $HoldsAt(f,t)$ to state that a fluent f holds at time t. $Happens(e,t)$ denotes that the event e happens at t. $Initiates(e,f,t)$ and $Terminates(e,f,t)$ define effects of events.[1] We use Δ to denote conjunctions of *Happens*-statements and γ to denote conjunctions of *HoldsAt*-statements. An *effect axiom* has the form $\gamma \Rightarrow \pi(e,f,t)$ where $\pi \in \{Initiates, Terminates\}$. A *precondition axiom* has the form $Happens(e,t) \Rightarrow \gamma$, saying that an event can only happen if condition γ holds.[2]

Planning in EC is abductive reasoning. (Shanahan, 2000) describes EC planning as follows: Consider Γ to be the initial world state, Γ' the goal state and Σ a set of action specifications. One is interested in finding a plan Δ such that:

$$CIRC[\Sigma; Initiates,\ Terminates,\ Releases] \wedge CIRC[\Delta; Happens] \wedge \Gamma \wedge \Omega \wedge EC \models \Gamma' \quad (1)$$

where *CIRC* denotes circumscription (Mueller, 2005) and Ω denotes uniqueness of names axioms.

Patkos and Plexousakis (2009) developed DECKT, an epistemic theory for EC. They show that the theory is sound and complete wrt. **T** system (Blackburn et al., 2001) of the possible world semantics.They introduce an epistemic *Knows*-fluent using nested reification. For example, $HoldsAt(Knows(\neg f),t)$ means that at time t the agent knows that f is false. *Knows*-fluents are released from inertia at all times in DECKT. DECKT also uses a fluent *KP(f)* which states that f is known persistently, i.e. *KP*-fluents are not released from inertia. DECKT states, that everything which is *KP*-known is also *Knows*-known:

$$HoldsAt(KP(\phi),t) \Rightarrow HoldsAt(Knows(\phi),t) \quad (2)$$

Reification allows for expressing so-called Hidden Causal Dependencies (HCD). HCDs are implications like $HoldsAt(KP(f \Rightarrow f'),t)$, expressing that it is known that if f is true then f' is also true. DECKT's axiom (3) states that a fluent can only be known if it is *KP*-known or if it is known through an implication.

$$HoldsAt(Knows(\phi),t) \Rightarrow HoldsAt(KP(\phi),t) \vee \left(HoldsAt\left(KP(\phi'),t\right) \wedge HoldsAt\left(KP(\phi' \Rightarrow \phi),t\right)\right) \quad (3)$$

3 EPISTEMIC PLANNING

A problem specification consists of a set of types $\mathcal{T}$, a set of objects O, a set of fluents $\mathcal{F}$, a set of action specifications $\mathcal{A}$, a set of goals $\mathcal{G}$ and a set of statements about the agent's initial knowledge I which may be incomplete. $\mathcal{T}$, O, $\mathcal{F}$, $\mathcal{A}$, $\mathcal{G}$ and I are finite and may be empty.

Types are *sorts* in an EC domain description. A PLIK type specification is e.g.:

```
(:types Door Room Robot)
```

[1] Throughout this text, all variables are universally quantified if not stated otherwise. Variables for events/actions are denoted by e, for fluents by f, for literals by l and for time by t. Reified fluent formulae ϕ do not contain quantifiers or predicates. Second order expressions do not occur.

[2] For more details concerning precondition axioms we refer to (Mueller, 2005).

Objects are elements of a certain sort:

```
(:objects corridor,livingRoom - Room
          d1 - Door wc,vc - Robot )
```

where the "-" is to be read as "element-of".
Fluents have a set of arguments of a certain type. An example for the fluent specification in PLIK is:

```
(:fluents
  hasDoor(Room,Door) inRoom(Robot,Room)
  opened(Door) isRobust(Robot) )
```

There may exist a sensing action for a fluent f. Therefore we define two deterministic events for each fluent, *senseT(f)* and *senseF(f)*.

$$\begin{aligned} &Initiates\,(senseT\,(f),KP(f),t)\\ &Terminates\,(senseT\,(f),KP(\neg f),t)\\ &Initiates\,(senseF\,(f),KP(\neg f),t)\\ &Terminates\,(senseF\,(f),KP(f),t) \end{aligned} \tag{4}$$

Goals are specified as follows:

```
(:goal (and (or [3] inRoom(wc, livingRoom) )
            (or [4] !inRoom(vc, livingRoom) ) ) )
```

where the numbers in square brackets represents the time limit until the literal must be known to hold. Its general form is conjunctive normal form (CNF):

```
(and (or [t_{l_1}]l_{l_1} ... [t_{l_n}]l_{l_n}) ...
     (or [t_{m_1}]l_{m_1} ... [t_{m_n}]l_{m_n}))
```

It translates to epistemic EC as follows:

$$\bigwedge_{i=1}^{m}\bigvee_{j=1}^{n} HoldsAt\left(Knows(l_{i_j}),t_{i_j}\right) \tag{5}$$

Initial Knowledge is provided as follows:

```
(:init  inRoom(wc,corridor) inRoom(vc,corridor)
        isRobust(vc) hasDoor(corridor, d1)
        hasDoor(livingRoom, d1) )
```

It is a set I of literals which are known to hold at time 0. They translate to epistemic EC as:

$$\bigwedge_{l_i\in I} HoldsAt\,(KP(l_i),0) \tag{6}$$

EC does not use negation-as-failure, so we also have to explicitly state what the agent does not know:

$$\bigwedge_{l_i\in I} l\neq l_i \Rightarrow \neg HoldsAt\,(KP(l),0) \tag{7}$$

where the l_i are these literals the agent knows.
Action Specifications. We illustrate the translation of action specifications with the following example:

```
(:action moveRoomToRoom
  :parameters (?robo - Robot ?door - Door ?from
               ?to - Room)
  :precondition (and
    inRoom(?robo,?from) !inRoom(?robo,?to)
    hasDoor(?from, ?door) hasDoor(?to, ?door)
    (or isRobust(?robo) opened(?door)))
  :effect
    (if (and opened(?door) inRoom(?robo, ?from))
     then (and  inRoom(?robo, ?to)
                !inRoom(?robo, ?from))))
```

It describes the conditional effect that if a robot executes this action it will end up in the target room if the door to the room is open. The precondition states that the agent will only consider this action if it knows that the door is open or that the robot is robust (so crashing against a closed door does not cause harm). It consists of a parameters section, a precondition specification and a conjunction of conditional effects. The parameters are variables (denoted with the preceding `?`) and quantified over the scope of their sort. An action's precondition is given in CNF:

```
(and (or l_{l_1} ... l_{l_n}) ... (or l_{m_1} ... l_{m_n}))
```

Let e be the action's name, then we have in EEC:

$$Happens\,(e,t)\Rightarrow\bigwedge_{i=1}^{m}\bigvee_{j=1}^{n} HoldsAt\left(Knows(l_{i_j}),t\right) \tag{8}$$

This states, that an event can only happen if its preconditions are known to hold. In terms of planning, it means that the planner will only consider actions in a plan when it *knows* that their preconditions hold.
A conditional effect of an action has the form:

```
(if (and l_1 ... l_k) then (and l_1 ... l_m))
```

It can be represented as a pair (E,C) where $C=\{l_1,\ldots,l_k\}$ is a set of condition literals and $E=\{l_1,\ldots,l_m\}$ is a set of effect literals. Even though we consider epistemic planning, the physical non-epistemic effects of actions are still valid. Thus, we have to add one non-epistemic effect axiom for each $l_j\in E$ to our theory:

$$\bigwedge_{l_i\in C} HoldsAt\,(l_i,t)\Rightarrow\pi(e,f_j,t) \tag{9}$$

where $l_i\in C$ are the condition literals, f_j are the fluents of the literals $l_j\in E$ and $\pi\in\{Initiates,Terminates\}$.
DECKT states that if all condition literals of a conditional effect are known to hold, then its effect literals are also known to hold. For every effect literal l_j in a

conditional effect of an action we add:

$$\bigwedge_{i=1}^{n} HoldsAt(Knows(l_i),t) \Rightarrow Initiates(e,KP(l_j),t) \quad (10)$$

Knowledge about an effect fluent is lost if a) at least one of the conditions is unknown and if b) there is no condition which is known not to hold and if c) the new truth value of the effect fluent is not already known. Thus, for every $l_j \in E$ we add the effect axiom:

$$\begin{aligned} &\left(\bigvee_{l_i \in C} \neg HoldsAt(Knows(l_i),t)\right) \wedge \\ &\left(\bigwedge_{l_i \in C} \neg HoldsAt(Knows(\neg l_i),t)\right) \wedge \\ &\neg HoldsAt(Knows(l_j),t) \\ \Rightarrow\ &Terminates(e,KP(l_j),t) \end{aligned} \quad (11)$$

DECKT defines how *knowledge about the effect of an action is obtained through knowledge about its condition*. This is useful, e.g. if for some reason sensing a door's open-state is not possible in certain rooms. In this case one can send the robot through the door, sense its open-state *after* the execution and *retroactively* infer the robot's location.

At the current state we only account for the case where one condition is unknown. If knowledge about the unknown condition is acquired later and the condition is revealed to hold, then knowledge about the effect is revealed indirectly.

Thus, for every effect literal $l_j \in E$ and for every potentially unknown condition literal $l_u \in C$ we have the following clause:

$$\begin{aligned} &\left(\bigwedge_{l_i \in C \setminus \{l_u\}} HoldsAt(Knows(l_i),t)\right) \wedge \\ &\neg HoldsAt(Knows(\neg l_u),t) \wedge \\ &\neg HoldsAt(Knows(l_j),t) \\ \Rightarrow\ &Initiates(e,KP(l_u \Rightarrow l_j),t) \end{aligned} \quad (12)$$

The event initiates an *implication*, stating that if l_u is true then l_j must also be true.

In order to *acquire knowledge about a condition through an effect*, the effect of an action must be known not to hold at the action's execution time, and there must be no condition which is known not to hold. If the effect later becomes known to hold, then all conditions of the actions must also hold. Thus, for each condition literal $l_j \in C$ and each effect literal $l_i \in E$, we add the following clause to our theory:

$$\begin{aligned} &\neg HoldsAt(Knows(l_i),t) \wedge \\ &\neg HoldsAt(\neg Knows(l_j),t) \wedge \\ &\neg HoldsAt(Knows(l_j),t) \\ \Rightarrow\ &Initiates(e,KP(l_j \Rightarrow l_i),t) \end{aligned} \quad (13)$$

where e is the action's name.

DECKT uses the predicate *KmAffect* to express that an event may affect a fluent. The predicate holds for each effect $l_j \in E$ of a conditional effect (C,E) of an event e if there is no condition which is not known not to hold:

$$\begin{aligned} &Happens(e,t) \wedge \bigwedge_{l_i \in C} \neg HoldsAt(Knows(\neg l_i),t) \\ &\Rightarrow KmAffect(e,f_j,t) \end{aligned} \quad (14)$$

where f_j is the fluent in the literal l_j.
An implication is terminated if one of the involved fluents may be affected.

$$\begin{aligned} &HoldsAt\left(KP(l \Rightarrow l'),t\right) \wedge \\ &KmAffect(e,f,t) \vee KmAffect(e,f',t) \\ \Rightarrow\ &Terminates\left(e,KP(l \Rightarrow l'),t\right) \end{aligned} \quad (15)$$

There are four more axioms we do not explicitly mention in this paper for brevity reasons. One states that *Knows*-fluents are always released from inertia. The epistemic truthness axiom says that everything which is known to hold does indeed hold. Another axiom determines how knowledge which is gained through an implication becomes persistent if the implication is terminated and the last axiom handles transitivity of implications which sometimes must be made explicit. We point the interested reader to (Patkos and Plexousakis, 2009) and refer to these two axioms as D^-.

Observations are a key element of epistemic theories. They provide the agent's basis for knowledge acquisition. In PLIK we use the keyword `:observation` to describe sensing effects. For example:

```
(:action senseInRoom
  :parameters (?robo - Robot ?room - Roomf)
  :observation (and inRoom(?robo, ?room) ) )
```

This action does not have a precondition or a physical effect and is thus a pure sensing action.The observations are provided as a conjunction `(and` $f_1 \ldots f_{|O|}$`)`. For each observed f_i we add the event axiom:

$$\begin{aligned} &Happens(e(\overline{v}),t) \Rightarrow \\ &Happens(senseT(f_i),t) \vee Happens(senseF(f_i),t) \end{aligned} \quad (16)$$

with $f_i \in \overline{v}$. Intuitively, if a sensing action happens, then either positive or negative sensing will happen.

Solving Epistemic Planning Problems. Given a planning problem description in PLIK, we generate an EEC representation as follows:

1. Objects, types and fluents are declared.
2. For each fluent we declare and specify *senseT* and *senseF* events Σ_s as described in (4).
3. Initial knowledge Γ is specified as described in (6), (7); goals Γ' are specified as described in (5).
4. Action specifications Σ are generated through (8), (9), (10), (11), (12), (13), (14) and (16).

The pair (Δ, Γ^*) is a *possible solution* to the planning problem with incomplete knowledge if the following holds:

$$CIRC[\Sigma \wedge \Sigma_s; \textit{Initiates, Terminates, Releases}] \wedge \quad (17)$$
$$CIRC[\Delta; \textit{Happens}] \wedge \Gamma \wedge \Gamma^* \wedge \Omega \wedge EC \wedge DECKT \models \Gamma'$$

where $DECKT = (2) \wedge (3) \wedge (15) \wedge (D^-)$ represents the domain-independent DECKT axioms which are not generated by our translation. Γ^* can be interpreted as a *possible world* in which the plan Δ is a solution to the planning problem, i.e. Γ^* is a conjunction of *HoldsAt* statements which can be seen as a condition under which Δ solves the planning problem. Figure 2 shows how Γ^* and Δ are included in the stable models generated by the ASP tools. The complete solution of the planning problem is the set of all n pairs (Δ_n, Γ_n^*) for which the entailment (17) holds. This solution can be interpreted as a conditional plan with the conditions represented by Γ_i^* and the actions to be executed under a certain condition by Δ_i. The execution of this plan then demands for an execution monitor which compares the possible worlds Γ_i^* with the agent's knowledge about the world. At execution time, the agent non-deterministically chooses a plan Δ_i and follows it as long as the condition Γ_i^* is not inconsistent with the agent's knowledge about the world. If Γ_i^* becomes inconsistent due to sensing actions, then the agent has to choose another branch (Δ_j, Γ_j^*) with a Γ_j^* that is consistent with the agent's knowledge about the world. If no such Γ_j^* exists, then the problem is unsolvable.

Implementation Issues. Lee and Palla (2012) propose to use Answer Set Programming to solve EC reasoning problems and show that their approach outperforms existing solutions like the DECreasoner by (Mueller, 2005). We make use of their f2lp tool to translate the EEC planning problem description into an ASP representation of the problem. We use the tool gringo (Gebser et al., 2011) to generate the Herbrand Models of the problem and claspD (Drescher et al., 2008) as the ASP solver. Unfortunately, not the f2lp-tool we use nor any other reasoner we know about supports reification. Therefore, we had to sacrifice completeness of our planner and define special predicates which translate as follows:

$$\begin{aligned}
&Knows(f,t) := HoldsAt(Knows(f),t)\\
&KnowsNot(f,t) := HoldsAt(Knows(\neg f),t)\\
&KP(f,t) := HoldsAt(KP(f),t)\\
&KPNot(f,t) := HoldsAt(KP(\neg f),t) \quad (18)\\
&ImpliesTT(f,f',t) := HoldsAt\left(KP(f \Rightarrow f'),t\right)\\
&ImpliesTF(f,f',t) := HoldsAt\left(KP(f \Rightarrow \neg f'),t\right)\\
&ImpliesFT(f,f',t) := HoldsAt\left(KP(\neg f \Rightarrow f'),t\right)\\
&ImpliesFF(f,f',t) := HoldsAt\left(KP(\neg f \Rightarrow \neg f'),t\right)
\end{aligned}$$

We define the persistence laws for the *KP* and *Implies* predicates equivalently to the persistence laws for the *HoldsAt* predicate in the DEC axiomatization and use special *Initiates* and *Terminates* predicates for each of the *KP* and *Implies* which follow the very same axioms as the original DEC *Initiates* and *Terminates* do. Contraposition and transitivity rules for the implication predicates is also implemented manually. Using these special predicates and the additional persistence and effect axioms for each predicate, our approach is sound. This is easy to see as none of our translation rules generates an implication which involves more than two fluents. However, it is not complete as we do not include HCD expansion as described in (Patkos, 2010), and we do not consider retroactive knowledge gain about effects if more than one condition is unknown.

4 EVALUATION

Reduction of the Planning Problem Specification. Looking at the translation of PLIK to non-epistemic EC, we find that through (9), for each conditional effect (C,E), $|E|$ effect axioms of the form $\gamma \Rightarrow \pi(e,f,t)$ are generated; one for each effect literal $l \in E$. Looking at the translation into the epistemic EC, we find that many additional epistemic effect axioms are generated: (10) generates $|E|$ effect axioms, (11) generates another $|E|$ axioms and (12), (13) generate $|E| \cdot |C|$ axioms each. Thus, each conditional effect (C,E) of an action specification demands specifying $|E|$ non-epistemic effect axioms and $|E| \cdot (2 \cdot |C| + 2)$ epistemic effect axioms.

For example, consider extending the above `moveRoomToRoom` action by adding only two more conditions (robot's battery must be full and robot must not be blocked) to the effect:

```
:effect (if (and
  opened(?door) batteryFull(?robo)
  !blocked(?robo) inRoom(?robo, ?from))
   then (and  inRoom(?robo, ?to)
              !inRoom(?robo, ?from))))
```

Then, the epistemic EC version of the action is more than 100 lines (11 000 bytes) of axioms which would be very circumstantial to implement manually.

Use case: Move-through-door. We implemented a use case with several robots moving through rooms and one central planning agent controlling the robots. Some robots are equipped with bumpers, so it is safe to send them through a door without knowing whether the door is open. The planning agent can only send a robot through doors if it knows that the door is open or that the robot has a bumper.[3]

We investigated the stable models which are generated by the reasoner and it turned out that the knowledge-level effects of actions are correctly handeled. Figure 2 shows how implications (I), plan (Δ), assumptions about the world (Γ^*), direct knowledge gain (K_d^+) as well as indirect knowledge gain (K_i^+) are successfully generated and modeled. The output is that of a simple scenario with two robots, `wc` and `vc`, where `vc` is robust and `wc` is not. The generated plan involves sending the robust `vc` through a door at $t = 0$, sensing its location at $t = 1$ and then, knowing that the door must be open at $t = 2$, safely sending `wc` through the door.

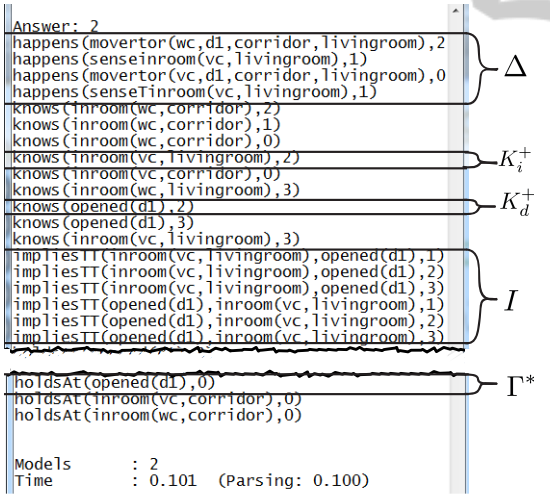

Figure 2: The output of the reasoner for the move-through-door example with two robots (vc,wc).

Use Case: Poisonous Liquid. We adopted the poisonous liquid-example from (Petrick and Bacchus, 2004). A thirsty agent has a liquid and does not know whether it is poisonous. He can perform a drink-action. If he drinks the liquid he will not be thirsty anymore but he will get poisoned if the liquid is poisonous. The agent can also pour the liquid on the lawn, and if the liquid is poisonous the lawn will be dead. Finally the agent can sense whether the lawn is alive. In the initial state, the agent only knows that he is thirsty and not poisoned. The goal is that the agent always knows that he is not poisoned and knows that in the end he is not thirsty. The result is exactly 1 stable model, because there is only one possible world in which the goal can be achieved: This is the world where the liquid is not poisonous. The plan considers that the agent first pours the liquid on the lawn, then senses whether the lawn is dead, and finally, if the lawn is not dead, drinks the liquid. For details concerning the implementation of this use case we refer the reader to (Eppe and Dylla, 2012).

[3]See the corresponding move-action in section 3.

5 CONCLUSIONS

In this paper we provide a method for the formalization of epistemic planning problems. The main contribution of our approach lies in the automated translation of planning problem specifications demanding complete knowledge about the world into planning problems which allow for incomplete knowledge. We show that this translation safes a problem designer the work of specifying additional $|E| \cdot (2 \cdot |C| + 2)$ knowledge-level effect axioms.To the best of our knowledge there is currently no other planning system which takes ordinary planning domains as input and automatically compiles them into epistemic planning domains, such that epistemic effects of actions can be exploited to compensate missing or broken sensors.

Our approach is sound but not complete wrt. DECKT and the possible worlds semantics of knowledge. To achieve completeness we have to introduce actions with non-deterministic effects, e.g. like tossing a coin. Another issue that we need to consider is knowledge acquisition about effects if more than one condition is unknown. This goes along with what (Patkos, 2010) calls HCD-expansion. However, this is hardly possible without true reification and we don't know of any reasoner which supports this.

ACKNOWLEDGEMENTS

We thank Theodore Patkos who was always happy to help us with details concerning DECKT and related work. Funding by the German Research Association (DFG) under the grants of International Research Training Group on Semantic Integration of Geospatial Information (IRTG SIGI) and SFB/TR8 Spatial Cognition is gratefully acknowledged.

REFERENCES

Bertoli, P., Cimatti, A., Lago, U. D., and Pistore, M. (2002). Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In *ICAPS Workshop on PDDL*.

Bertoli, P., Cimatti, A., Pistore, M., Roveri, M., and Traverso, P. (2001). MBP : a Model Based Planner. In *IJCAI Proceedings*.

Blackburn, P., de Rijke, M., and Venema, Y. (2001). *Modal Logic*. Cambridge University Press.

De Giacomo, G. and Levesque, H. (1998). An Incremental Interpreter for High-Level Programs with Sensing. In *Working Notes of the 1998 AAAI Fall Symposium on Cognitive Robotics*.

Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., and Schaub, T. (2008). Conflict-Driven Disjunctive Answer Set Solving. In *International Conference on Principles of Knowledge Representation and Reasoning*.

Eppe, M. and Dylla, F. (2012). An Epistemic Planning System Based on the Event Calculus. Technical Report 033-11/2012, University of Bremen, Bremen.

Gebser, M., Kaminski, R., König, A., and Schaub, T. (2011). Advances in gringo series 3. In *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning*.

Kowalski, R. (1986). A Logic-based calculus of events. *New generation computing*, 4:67–94.

Lee, J. and Palla, R. (2012). Reformulating the Situation Calculus and the Event Calculus in the General Theory of Stable Models and in Answer Set Programming. *Journal of Artificial Intelligence Research*, 43:571–620.

Moore, R. (1985). A formal theory of knowledge and action. In Hobbs, J. and Moore, R. C., editors, *Formal theories of the commonsense world*. Ablex, Norwood, NJ.

Mueller, E. (2005). *Commonsense reasoning*. Morgan Kaufmann.

Patkos, T. (2010). *A Formal Theory for Reasoning About Action , Knowledge and Time*. PhD thesis, University of Crete - Heraklion Greece.

Patkos, T. and Plexousakis, D. (2009). Reasoning with Knowledge , Action and Time in Dynamic and Uncertain Domains. In *IJCAI Proceedings*, pages 885–890.

Petrick, R. P. A. and Bacchus, F. (2004). Extending the knowledge-based approach to planning with incomplete information and sensing. In *ICAPS Proceedings*.

Reiter, R. (2001). *Knowledge in action: Logical foundations for specifying and implementing dynamical systems*. MIT Press.

Scherl, R. and Levesque, H. J. (2003). Knowledge, action, and the frame problem. *Artificial Intelligence*.

Shanahan, M. (2000). An abductive event calculus planner. *The Journal of Logic Programming*, pages 207–240.

Thielscher, M. (1998). Introduction To The Fluent Calculus. *Linköping Electronic Articles in Computer and Information Science*, 3(14).

Thielscher, M. (2005). FLUX : A Logic Programming Method for Reasoning Agents. *Theory and Practice of Logic Programming*, 5(4-5).