Narrative based Postdictive Reasoning for Cognitive Robotics

Manfred Eppe and Mehul Bhatt

University of Bremen

{meppe, bhatt}@informatik.uni-bremen.de

Abstract

Making sense of incomplete and conflicting narrative knowledge in the presence of abnormalities, unobservable processes, and other real world considerations is a challenge and crucial requirement for cognitive robotics systems. An added challenge, even when suitably specialised action languages and reasoning systems exist, is practical integration and application within large-scale robot control frameworks.

In the backdrop of an autonomous wheelchair robot control task, we report on application-driven work to realise postdiction triggered abnormality detection and re-planning for real-time robot control: (a). Narrative-based knowledge about the environment is obtained via a larger smart environment framework; and (b) Abnormalities are postdicted from stable-models of an answer-set program corresponding to the robot's epistemic model. The overall reasoning is performed in the context of an approximate epistemic action theory based planner implemented via a translation to answer-set programming.

Introduction

Researchers in the field of reasoning about action and change have interpreted *narratives* in several ways, differing in the richness of their semantic characterisation and ensuing formal properties (Miller and Shanahan 1994; Pinto 1998),(Mueller 2007),(McCarthy and Costello 1998; McCarthy 2000). For instance, within the context of formalisms such as the situation calculus and event calculus, narratives are interpreted as "*a sequence of events about which we may have incomplete, conflicting or incorrect information*" (Miller and Shanahan 1994; Pinto 1998). The interpretation of narrative knowledge in this paper is based on these characterisations, especially in regard to the representation and reasoning tasks that accrue whilst modelling the perceptually grounded, narrativised epistemic state of an autonomous agent. In this paper, we are especially concerned with large-scale cognitive robotics systems where high-level symbolic planning and control constitutes one of many AI sub-components guiding low-level control and attention tasks (Suchan and Bhatt 2012).

Perceptual Narratives and Postdiction

Perceptual narratives correspond to declarative models of visuo-spatial, auditory, haptic and other observations in the real world that are obtained via artificial sensors and / or human input (Bhatt, Suchan, and Schultz 2013). From the formal viewpoint of commonsense reasoning, computational modelling and reasoning with perceptual narratives encompasses logics of *space, actions, and change* (Bhatt 2012).[1]

Declarative models of perceptual narratives can be used for interpretation, plan generation, and control tasks in the course of assistive technologies in everyday life and work scenarios, e.g., in domains such as human activity recognition, semantic model generation from video, ambient intelligence and smart environments (e.g., see narrative based models in (Hajishirzi et al. 2012; Hajishirzi and Mueller 2011; Mueller 2007; Bhatt and Flanagan 2010; Dubba et al. 2012; Bhatt, Suchan, and Schultz 2013)). The focus of this paper is on particular inference patterns and an overall control architecture for online / incremental reasoning with narrative knowledge from the viewpoint of plan generation and explanation. We are especially interested in *completion of narrative knowledge* by inferring perceptual abnormalities and causes of perceived changes in the agent's world.

Explanation by postdictive reasoning within the framework of perceptual narratives can be the basis of explaining phenomena or properties perceived via sensory devices (Poole, Goebel, and Aleliunas 1987; Miller and Shanahan 1994). Given perceptual narratives available as sensory observations from the real execution of a system, the objective is often to assimilate / explain them with respect to an underlying domain / process model and an approach to derive explanations. The *abductive explanation* problem can be stated as follows (Kakas, Kowalski, and Toni 1992): given theory T, observations G, find an explanation $\triangle$ such that: (a). $T \bigcup \triangle \vDash G$; and (b). $T \bigcup \triangle$ is consistent. In other words, the observation follows logically from the theory extended given the explanation. Amongst other things, this can be used to identify abnormalities in a narrative, which may in turn affect subsequent planning and overall (agent) control behaviour.

---

[1]This paper does not directly address spatial representation and reasoning. Instead, the focus here is on *action and change*.

Narrative-based Incremental Robot Control

Our application of narrative-based incremental agent control is based on plan *monitoring*, and combining it with a mechanism for *explanatory reasoning*: if a monitored world property changed unexpectedly, then our system postdicts possible explanations that describe what may have happened that caused this change. A common paradigm used in the planning community is *strong* planning, see e.g. (Bertoli et al. 2002). A strong plan guarantees that the goal is achieved, no matter how the (partially unknown) world is. However, this paradigm is not appropriate when considering abnormalities: it may always happen that a plan does not succeed due to unexpected system failures. As such, we use an *incremental weak planning* approach, and interleave the planning with plan execution. A weak plan must not guarantee that the goal will be achieved, it must only show possibilities to achieve a goal. The overall system is implemented such that as soon as one weak plan is found the system starts acting. This weak plan is then extended (i.e. made "stonger") during plan-execution. Further, sensing results which are obtained during plan-execution are integrated in an online manner, and the search space is pruned accordingly during plan execution.

The narrative-driven explanation and control framework of this paper is built on a planning formalism called *h-approximation* ($\mathcal{HPX}$) that is incomplete but sound wrt. the possible-world semantics of knowledge (Eppe, Bhatt, and Dylla 2013). For $\mathcal{HPX}$ a corresponding planning system has been implemented via translation to an answer-set program. The paper extends this planning system with new features such that:

- it is capable of incremental online-planning, and
- it allows for abductive explanatory reasoning during plan execution.

The paper presents an overview of the basic offline h-approximation, and describes the extensions of the proposed online version together with a detailed architecture of the overall control approach.[2] We also bring forth the application guided motivations of our work by illustrating a real-time control task involving an autonomous wheelchair robot in a smart home environment. Finally, we present ongoing work aimed at integrating and delivering our online planner as a part of the experimental cognitive robotics framework ExpCog (Suchan and Bhatt 2012).

Approximate Epistemic Planning as ASP

We choose $\mathcal{HPX}$ as the theoretical foundation for our framework because it has native and elaboration tolerant support for postdictive reasoning along with a low computational complexity (the plan existence problem is in NP). For alternative $\mathcal{PWS}$ based formalisms, plan existence is Σ_2^P-complete, e.g. (Baral, Kreinovich, and Trejo 2000).

---

[2]The extended planning system uses the online ASP reasoner *oclingo* (Gebser et al. 2011a) that dynamically adopts its knowledge base according to sensing results.

To the best of our knowledge, no other implemented formalism supports postdiction in this complexity class. The support for postdiction is crucial to realize abnormality- and explanation-based error-tolerance in robotic systems: if sensing reveals that the effect of an action is not as intended, then postdiction can be used to abduce and explain the reason for the failure. This is a partial solution to the *qualification problem*: it is not possible to model all conditions under which an action has the intended effect. In this work we perform abnormality reasoning in an epistemic open-world sense. That is, we do not further describe (and circumscribe) abnormalities but rather use generic abnormality predicates as qualifications (negative conditions) of actions.

$\mathcal{HPX}$ is formalized and implemented in ASP: a problem specification is specified in PDDL-like syntax and is then translated to an Answer-Set Program (Gelfond and Lifschitz 1988) via a set of translation rules. The stable models of the generated Logic Program can be interpreted as conditional plans. The fact that $\mathcal{HPX}$ is implemented as ASP and not in a procedural programming language like c++ makes it simple to extend the formalism and its semantics on a logical level. For this paper, ASP solvers like *oclingo* (Gebser et al. 2011a) providing incremental and online problem solving-capabilities are relevant. Online problem solving makes it possible to dynamically add rules to a Logic Program. That is, the solver is in running in a loop and constantly awaiting new input via extra logical means. Whenever new rules are received, the solver tries to find new stable models according to the updated program.

Planning Problem Specification. A problem domain is specified using a syntax similar to the planning domain definition language (PDDL): (`:init` l^{init}) represents initial knowledge about a literal l^{init}. (`oneof` $l_1^{oo} \ldots l_n^{oo}$) describes initial exclusive-or-knowledge. (`:action` a `:effect when (and` $l_1^c \ldots l_n^c$) l^e) is called an *effect proposition* (EP). It represents the conditional effect of an action a, that if the condition literals $l_1^c \ldots l_n^c$ hold, then the effect l^e will also hold. (`:action` a `:observe` f) is a *knowledge proposition*. It represents that an action a will sense the value of a fluent f. (`:action` a `:executable (and` $l_1^{ex} \ldots l_n^{ex}$)) is an *executability condition*. An action is only executable if literals $l_1^{ex} \ldots l_n^{ex}$ are known to hold. Finally, (`:goal weak` l^g) is used to state weak goals, i.e. goals which are satisfied by a plan such that a desired property l^g is known to be achieved in at least one leaf of the search tree. We do not consider strong goals (a goal that must be known to hold in all leafs) because we consider an open world where it is impossible to model all qualifications of an action. Hence an action can always fail, and it is impossible to predict that a goal is achieved in all leafs of a transition tree.

Planning Problem Formalization. The h-approximation for a planning problem $\mathcal{P}$ consists of two parts:

- Σ_{hapx}: a set or rules representing a foundational domain-independent theory
- $\mathcal{P} \overset{\mathbf{T}}{\longmapsto} \Sigma_{world}$: the translation of a planning problem $\mathcal{P}$ into a domain-specific theory Σ_{world} using a set of translation rules $\mathbf{T}$.
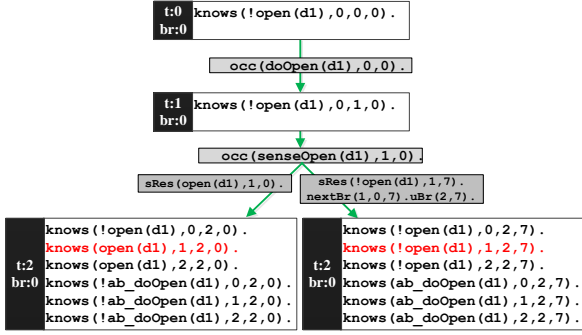
```
t:0   knows(!open(d1),0,0,0).
br:0

      occ(doOpen(d1),0,0,0).

t:1   knows(!open(d1),0,1,0).
br:0

      occ(senseOpen(d1),1,0).

sRes(open(d1),1,0).        sRes(!open(d1),1,7).
                          nextBr(1,0,7).uBr(2,7).

      knows(!open(d1),0,2,0).    knows(!open(d1),0,2,7).
      knows(open(d1),1,2,0).     knows(!open(d1),1,2,7).
t:2   knows(open(d1),2,2,0).  t:2  knows(open(d1),2,2,7).
br:0  knows(!ab_doOpen(d1),0,2,0).  br:0  knows(ab_doOpen(d1),0,2,7).
      knows(!ab_doOpen(d1),1,2,0).    knows(ab_doOpen(d1),1,2,7).
      knows(!ab_doOpen(d1),2,2,0).    knows(ab_doOpen(d1),2,2,7).
```

Figure 1: Example transition tree for $\mathcal{HPX}$

The resulting Logic Program, denoted by LP($\mathcal{P}$), is the conjunction $\Sigma_{hapx} \cup \Sigma_{world}$.

The following are the main predicates used in the ASP formalization of the $\mathcal{HPX}$:

- $occ(a, t, b)$ denotes that action a occurs at step t in branch b.
- $apply(ep, t, b)$ denotes that an effect proposition ep is applied at step t in branch b.[3]
- $sRes(l, t, b)$ denotes that the literal l is potentially sensed at step t in branch b.
- $knows(l, t, t', b)$ states that at step t' in branch b it is known that l holds (or did hold) at step t (with $t \leq t'$). That is, $\mathcal{HPX}$ does not only consider knowledge about the present state but also about the past.
- $nextBr(t, b, b')$ denotes that sensing happened at step t in branch b, resulting in a child-branch b'.
- $uBr(t, b)$ denotes that branch b is a valid branch at step t. Actions can only be executed when a branch is valid.
- $goal(l)$ denotes a (weak) goal for a literal l.

Example: Consider the following action specification:

```
(:action doOpen :parameters (?d - Door)
        :effect when ¬ab_doOpen (open ?d))
```

This represents an action where a door will be open if there is no abnormality. This specification is translated into an ASP formalization via the translation rules (T6a-c) presented in (Eppe, Bhatt, and Dylla 2013). For instance, translation rule (T6a) generates:

$$knows(open(D), T+1, T1, BR) \leftarrow$$
$$apply(doOpen(D)_0, T, BR),$$
$$knows(\neg ab\_doOpen(D), T, T1, BR), T1 > T.$$

That is, if the 0-th EP of the action `doOpen(D)` is applied at `T`, and at `T1>T` it is known that at `T` there is no abnormality then at `T1` it is known that after the action occurrence (at `T+1`) the door is open. Similarly, translation rule (T6c) in (Eppe, Bhatt, and Dylla 2013) generates:

$$knows(ab\_doOpen(D), T, T1, BR) \leftarrow$$
$$knows(\neg open(D), T+1, T1, BR),$$
$$apply(doOpen(D)_0, T, BR), T1 > T.$$

---

[3]In $\mathcal{HPX}$, actions are partitioned in EP to simplify reasoning with concurrency. Whenever $occ(a, t, b)$ and ep is an effect proposition of a, then $apply(ep, t, b)$

This reflects *negative postdiction*: If the open-action was executed at step `T` and at a step `T1` the effect of the open-action is known not to hold at `T+1`, then it is known that there must have been an abnormality at `T`. A similar *positive postdiction* rule is generated by (T6b) which produces knowledge that there is no abnormality if it is known that the effect of the action holds after execution but not before execution.

Consider a sensing action specification

```
(:action senseOpen :parameters (?d - Door)
        :observe (open ?d))
```

This generates knowledge as follows (see Figure 1): Assume the `drive`-action occurs in the initial node ($t = 0$, $br = 0$) in the transition tree. `senseOpen` is applied after `doOpen`, i.e. at step 1 in branch 0: i.e. $occ(senseOpen(d1), 1, 0)$. Then the positive sensing result is associated with the original branch 0: $(sRes(open(d1), 1, 0))$. The negative result is associated with a child branch, e.g. 7:[4] $sRes(\neg open(d1), 1, 7)$. Further, $nextBr(1, 0, 7)$ is generated to reflect that branch 7 is generated as a child of branch 0 at step 1. Finally, $uBr(2, 7)$ is indicates that branch 7 is valid from step 2 on, and hence the planner may consider actions to occurr in branch 7.

Narrativised Online Robot Control

The original h-approximation formalism and planning system is designed for offline problem solving. That is, a conditional plan is generated and the projected future world states are checked for whether a predefined goal holds. In this work, we extend h-approximation such that it is capable of online planning and abductive explanation. We also define several measures to assess the quality of a plan (e.g. robustness wrt. to unknown contingencies). A key feature of the h-approximation is the support for postdiction; we use postdiction to find explanations of why an action did or did not succeed. We propose to model actions such that the non-existence of an abnormality is a condition for the action to succeed. After executing the action, sensing can be applied to verify whether an action succeeded, and thus to postdict whether there was an abnormality. Identified abnormalities can then be used, for instance, in support of other kinds of reasoning or control tasks.

I. An Extended Online h-approximation

The overall online h-approximation architecture is depicted in Fig. 2: it consists of an online ASP solver and a controller, which serves as interface to human input devices and the robotic environment.

The complete LP to be solved is the conjunction of the LP translation of the domain specification $\mathcal{D}$, the goal specification $\mathcal{G}$ and an execution narrative $\mathcal{N}$: $LP(\mathcal{P}) = LP(\mathcal{D}) \cup LP(\mathcal{G}) \cup LP(\mathcal{N})$. Here, the execution narrative contains information about which actions were actually executed and which sensing results were actually obtained.

---

[4]The number of the child branch is randomly generated via a choice rule.
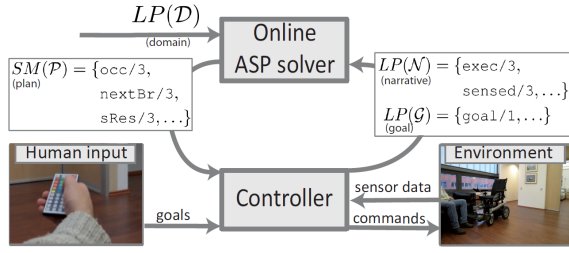
Figure 2: System Architecture

Once a stable model $SM(\mathcal{P})$ is found, it is sent to the controller which interprets this as a conditional plan and starts to execute it. It reports the execution narrative $\mathcal{N}$ back to the solver. The solver adopts the search space according to this information and refines / expands the plan accordingly. The updated stable models are thereupon sent to the controller again which acts accordingly. The loop is repeated until the goal is achieved or the problem becomes unsolvable.

Online Controller. We implement a controller which communicates new goals, sensing results and execution statements to the solver. It is responsible for the plan execution and the communication with actuators and sensors. Once an action is executed, the planner has to commit to this action, i.e. it must always consider the occurrence of this action. This mechanism is implemented with the following rule:

$$occ(A, t, B) \leftarrow \qquad\qquad\qquad (1)$$
$$exec(A, t, B), a(A), uBr(t, B).$$

where $exec(a, t, b)$ represents that an action a is executed at step t in branch b. The controller sends this information to the solver when it starts to execute the action. Once an action is initiated its execution will not be aborted.

The following choice rule generates plans.

$$\{occ(A, t, B) : a(A)\} \leftarrow$$
$$uBr(t, B), notGoal(t, B), \qquad (2)$$
$$not\ exec(A', t, B) : a(A').$$

where $notGoal(t, b)$ denotes that the goal is not yet achieved in branch b at step t. This prunes the search space because actions are only considered if the goal is not yet achieved at that node. Finally, $not\ exec(A', t, B)$ causes the solver not to generate any actions at step t if another action was already physically executed at that step.

Real-World sensing results are communicated from the controller to the solver in terms of *sensed* atoms. These are integrated into the agent's knowledge state by disabling the effect of projected sensing results which do not coincide with the actual sensing:

$$sRes(F, t, B) \leftarrow occ(A, t, B), hasKP(A, F),$$
$$not\ knows(\neg F, t, t, B), not\ sensed(\neg F, t). \qquad (3)$$
$$sRes(\neg F, t, B') \leftarrow occ(A, t, B), hasKP(A, F),$$
$$not\ kw(F, t, t, B), nextBr(t, B, B'), not\ sensed(F, t). \quad (4)$$

$hasKP(a, f)$ denotes that an action a has a knowledge proposition concerning a fluent f (i.e. it will sense f).

Whenever such an action occurs, the positive sensing result is always projected to the original branch, while the negative result is projected on a child branch given through $nextBr$. Projected sensing results are only valid if they do not contradict the actual sensing results (implemented by $not\ sensed$ statements).

Also, when receiving the actual sensing value, we have to take care that nodes which were valid in the projected search tree become invalid when the sensing contradicts the projections. The following rule implements that the original branch (where the pos. fluent was projected) becomes invalid if the sensing was negative ($\neg$ F) and the child branch becomes invalid if the sensing was positive:

$$brInvalid(t, B) \leftarrow sensed(\neg F, t),$$
$$occ(A, t, B), hasKP(A, F).$$
$$brInvalid(t, B') \leftarrow sensed(F, t), \qquad\qquad (5)$$
$$occ(A, t, B), hasKP(A, F), nextBr(t, B, B').$$

The information about invalid nodes is used for the generation of child branches. That is, a branch does not persist if it is invalid (6a) and it is also not created if invalid (6b).

$$uBr(t + 1, B) \leftarrow uBr(t, B), not\ brInvalid(t, B). \qquad (6a)$$
$$uBr(t + 1, B') \leftarrow nextBr(t, B, B'), not\ brInvalid(t, B'). \qquad (6b)$$

Incremental Planning Horizon Extension In online ASP solving, a single integer iterator (we use t) is incremented continuously until a solution is found.[6] This is sensible for quickly finding a first solution for a planning problem, as it guarantees that if a plan is found then it is minimal in length. Also, this plan is usually found very quickly because the search space is relatively small in the beginning.

However, as we perform weak planning, it may well be the case that the first found plan does not lead to the goal in practice. Therefore the planning-horizon is constantly incremented while the robot is executing the plan. That is, the plan is expanded to consider more contingencies while the robot is already acting.

Exogenous Actions (EA) are actions that occur but which the planning agent can not control. These actions can not be planned for as it is the case for *endogenous* actions, i.e. actions which can be executed by the controller. In our framework, we restrict exogenous actions in that they must have disjoint effect literals. This is necessary to avoid unwanted side-effects on knowledge which occur due to postdiction. Apart from that, we generate the ASP formalisation of an EA a as usual with $\mathcal{HPX}$ translation rules but flag it as exogenous by generating the fact $ea(a)$. In the context of Smart

---

[6]Incremental problem solving is realized by splitting a LP up into three parts: $\#base$, $\#cumulative$ and $\#volatile$. The $\#base$ part is an ordinary Logic Program while $\#cumulative$ and $\#volatile$ contain the iterator which expands the problem horizon. With each incrementation a new "slice" of the Logic Program is grounded and added to the set of rules. Incrementation takes place until a solution is found or up to a certain limit, depending on the configuration of the solver.
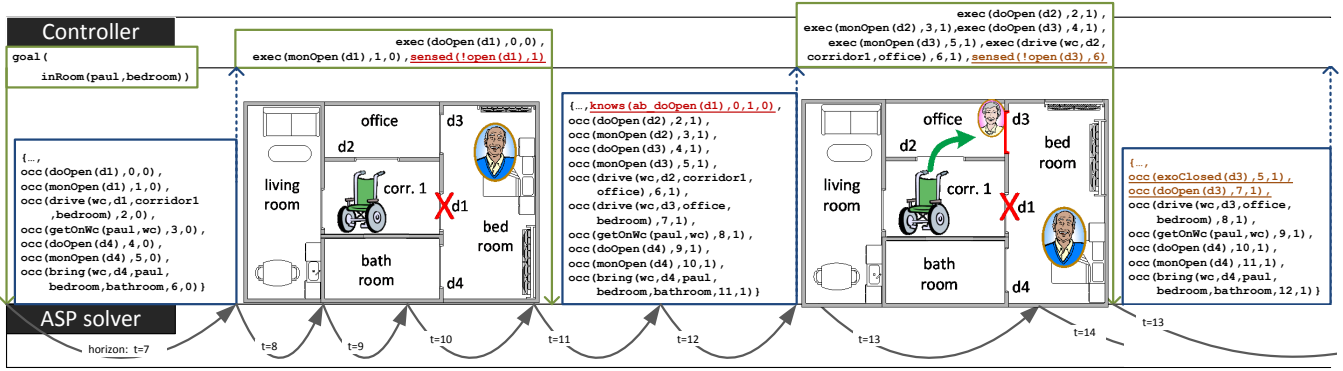
Figure 3: Communication between controller and ASP solver in Smart Home scenario[5]

Homes, the motivation behind to considering EA in planning is that "external" human agents often intuitively know what to do in a certain situation: For instance, if an autonomous wheelchair approaches a person and if the person needs this wheelchair, then it will "automatically" sit down on it. If human reaction is less automatic, then exogenous actions can often still be triggered by sending appropriate messages to human agents. For instance, one can model an exogenous action to fix an abnormality: The controller will notify external maintenance personnel about an abnormality and this should trigger fixing. Note that sensing is also allowed as exogenous action.

Though exogenous actions may lead to solutions which would not be found otherwise, the planner should first try to find a plan that does not contain exogenous actions. Limiting the number of exogenous actions is realized by the following rules:

$$maxExo(N,t) \leftarrow N = @mod(t,\mathtt{n}). \tag{7a}$$

$$numExo(N,t) \leftarrow N = \{occ(A,\_,\_) : ea(A)\}. \tag{7b}$$

$$\leftarrow maxExo(N,t), numExo(M,t), M > N. \tag{7c}$$

Instead of defining an absolute limit, we make the number of allowed exogenous actions dependent on the planning horizon: $@mod(t,n)$[7] returns the modulo of t and a constant n (1st rule), and determines the number of EA that may happen in a certain planning horizon. The second rule counts the number of exogenous actions and the integrity constraint (third rule) disables stable models where the number of exogenous actions is higher than allowed.[8]

Explanation Where a certain world property may change unexpectedly, it is useful to monitor this property continuously to make sure that the correct value of this property is always known. For instance, we may open a door and then send a robot through the door. However, we never know

whether the door was accidentally closed by another (human) agent in the meanwhile.

In our framework, unexpected change of world properties is modeled by explanation. We apply the usual inertia laws and consider unexpected change with abductive explanatory reasoning: If a world property changes unexpectedly, then our framework adds the updated knowledge to the domain model indirectly, by considering candidates for exogenous actions that may have caused this change. We implement this explanation mechanism as follows:

$$\begin{aligned} \mathit{0}\{exoHappened(A,t,BR) \\ : hasEP(A,EP) : hasEff(EP,\overline{L}) : ea(A)\} &\leftarrow \\ knows(L,t,t,B), sensed(\overline{L},t+1) \\ occ(A,t,BR) &\leftarrow exoHappened(A,t,BR). \end{aligned} \tag{8}$$

If it is known that at step t a literal L holds, but it is sensed that at $t+1$ the complement, $\overline{L}$ holds, then an exogenous action can have happened that has set L. Note that exogenous actions are only used for explanation if there occurred no endogenous action which may also have set the value of concern: The h-approximation has the restriction that no two actions with the same effect literal may happen concurrently. Therefore, if an endogenous action with the respective effect literal has been executed, an exogenous action with the same effect literal will not be considered for explanation. Further, explanation relies on the closed world assumption that all actions which can possibly occur are modeled in the domain, and that all exogenous actions have disjoint effect literals. Without these requirements it may happen that wrong beliefs are produced: If there are multiple actions which could explain an unexpected sensing result, then not all explanations will be true. If the explanation is wrong, and if the action which is used in the explanation has a condition, then false knowledge about these conditions could be postdicted. An alternative to the closed-world assumption is to restrict exogenous actions to have only one effect literal and no conditions. In that case, even though explanations about the occurrence of actions may be wrong they do not have side-effects on knowledge.

---

[7]The clingo family of ASP solvers (Gebser et al. 2012) support the definition of *lua* functions which can be used for simple auxiliary computation tasks.

[8]The integrity constraint appears in the $\#volatile$ part of the program, the other two rules in the $\#cumulative$ part.

Monitoring By monitoring we refer to continuous observation of a world property. As a methodological solution to represent monitoring in the domain specification we suggest to model pseudo-physical effects: If a sensor can monitor the value of a fluent f, then we add the physical effect (mon_f) to the action specification. For example, the following represents monitoring the open-state of a door:

```
(:action monOpen
 :parameters (?d – Door)
 :effect (mon_open ?d)
 :observe (open ?d))
```

Now, to model a "safe" drive-action where a door's open state is always known before passing it we add the precondition that the open-state of a door is monitored:

```
(:action drive
   :parameters (?robo – Robot ?door – Door
      ?from ?to – Room)
   :precondition (and (mon_open ?d)
      ⋮
```

II. Assessing Plan Quality

So far we have considered "raw" weak plans. These plans may still not be very appropriate in practice, e.g. because they contain cycles, are unlikely to lead to the goal or involve many exogenous events.

We use several optimization criteria to asses the quality of plans. An optimal plan should i) contain few exogenous actions ii) achieve the goal for as many contingencies as possible and iii) it should be possible to define soft-constraints or *maintenance goals* which must hold whenever possible. Finally, iv) the number of actions should be minimal. (With the priority of these criteria in the given order.)

Plan Strength The plan strength reflects for how many contingencies, i.e. unknown world properties, the goal is solved.

Definition 1 (Strength of a plan). *Given a plan p and a planning problem $\mathcal{P}$. Let $n_l(p, \mathcal{P})$ be the number of leafs of the search tree and $n_g(p, \mathcal{P})$ the total number of leafs in which the goal is achieved, then the strength s of p wrt. $\mathcal{P}$ is $s(p, \mathcal{P}) = \frac{n_g(p, \mathcal{P})}{n_l(p, \mathcal{P})}$*

In the logic program, the strength for each level t of the conditional plan is determined as follows:

$$
\begin{aligned}
leafs(L, t) &\leftarrow L = \{uBr(t, B)\}. \\
goals(G, t) &\leftarrow G = \{wGoal(t, B) : uBr(t, B)\}. \\
strength(S, t) &\leftarrow \\
& S = @div(G * 100, L), goals(G, t), leafs(L, t).
\end{aligned}
\tag{9}
$$

where L is the number of leafs, G is the number of leafs in which the goal is achieved and S is the plan strength. *oclingo* does only support integer numbers, so G is multiplied by a factor 100 and then divided by L. $wGoal(t, b)$ denotes that all weak goals are achieved in the respective node.

Maintenance Goals A maintenance goal is a soft-constraint which should hold as often as possible. The more nodes a search tree has where a maintenance goal is fulfilled, the higher is its quality. We call the corresponding assessment measure the *m-value* of a plan p wrt. a planning problem $\mathcal{P}$:

Definition 2 (m-value of a plan). *Given a plan p and a planning problem $\mathcal{P}$. Let $n_n(p, \mathcal{P})$ be the number of nodes of the search tree. Let $m_1, \ldots, m_n$ be maintenance goals. Then the m-value m of p wrt. $\mathcal{P}$ is $m(p, \mathcal{P}) = \frac{\sum_{m_i} n_{m_i}(p, \mathcal{P})}{n_n(p, \mathcal{P})}$, where n_{m_i} is the number of nodes in which a maintenance goal m_i holds.*

In terms of Logic Programming, the *m-value* wrt. a planning horizon t is obtained as follows:

$$
\begin{aligned}
nodes(N, t) &\leftarrow N = uBr(T, B) : s(T) : br(B). \\
mSum(M, t) &\leftarrow M = \{knows(L, t, t, B) : \\
& \qquad\qquad uBr(t, B) : mGoal(L)\}. \\
mVal(V, t) &\leftarrow V = @div(M * 100, N), \\
& \qquad mSum(M, t), nodes(N, t).
\end{aligned}
\tag{10}
$$

where $mGoal(l)$ atoms denote maintenance goals for a literal l.

Applying plan quality measures State-of-the-art ASP solvers like *clingo* (Gebser et al. 2011b) offer optimization statements to select an optimal stable model among the entire answer set. For example, the following statements cause an ASP solver to select one stable model with the minimal number of exogenous actions, maximal strength, maximal m-value and minimal number of actions (with descending priority):

$$
\begin{aligned}
&\#minimize[numExo(A, t) = A@4] \\
&\#maximize[strength(S, t) = S@3] \\
&\#maximize[mVal(M, t) = M@2] \\
&\#minimize[act(A, t) = A@1]
\end{aligned}
\tag{11}
$$

Unfortunately there is currently no ASP solver available which supports both incremental online problem solving and optimization statements.[9] As long as this is not implemented, selecting the plan with the highest quality has to be done by the controller, while the ASP solver can only perform the assessment of the plans. However, this is currently not implemented in our prototype.

Case Study: Abnormality Aware Wheelchair Robot

Our framework is integrated into a larger assistance system in a Smart Home environment, namely the Bremen Ambient Assisted Living Lab (BAALL) (Krieg-Brückner et al. 2010). The environment has at its disposal many actuators and sensors such as automatic doors, a smart TV, and also an autonomous robotic wheelchair. A typical (simplified) use case in this environment is illustrated in Fig. 3:

---

[9]However, this is currently being worked on and a version of *oclingo* which supports optimization will be released in near future (*Source*: personal conversation with Torsten Schaub, University of Potsdam).

Paul is in the bedroom and wants to get to the bathroom. This goal is sent from the controller to the ASP solver, and the planning starts. The solver finds the first plan with a horizon of 7 steps. The plan is sent to he controller and execution starts. Door d1 is opened and its open-state is verified by monitoring its open-state. It turns out that the door actually is not open and an abnormality is postdicted: $knows(ab\_doOpen(d1), 0, 1, 0)$. While these first actions were executed, the solver already incremented the planning horizon up to 10. However, to find an alternative this is not sufficient: the route via the office requires 12 steps in total, and accordingly the planning horizon must be at least 12 as well. This causes plan execution to be interrupted until the horizon is expanded. When the new plan is found, execution continues: Doors d2 and d3 are opened, and their open-states are monitored. However, while the wheelchair is driving through d2, Mary accidentally closes d3. This is immediately reported to the solver: $sensed(\neg open(d3), 6)$. It interrupts horizon extension to 14, and instead finds an explanation for the closed door – $occ(exoClosed(d3), 5, 1)$ – with a horizon of 13. It adopts the plan to the new situation, which considers that d3 has to be opened again. Thereafter the rest of the plan can be executed.

Related Work

The present work copes with planning, dynamic plan repair, abductive explanation and postdiction under incomplete knowledge and sensing actions. Therefore we are interested in other frameworks which have similar features. There are many action-theoretic frameworks and implementations such as the Event Calculus Planner by Shanahan (2000), but these often assume complete knowledge about the world and have no semantics that cover sensing actions.

We are interested in formalizations and implementations that cover incomplete knowledge, as found in the literature from the *contingent planning* community (e.g. CFF (Hoffmann and Brafman 2005) or MBP (Bertoli et al. 2001)). However these PDDL-based approaches are usually designed for offline-usage and hence not suitable for control tasks as illustrated in our case study. In addition, these approaches are usually based on some form of a $\mathcal{PWS}$ formalization and hence have a higher complexity than $\mathcal{HPX}$ (e.g. the plan-existence problem for $\mathcal{A}_k$ with $\mathcal{PWS}$-semantics is Σ_2^P complete).

PROSOCS (Bracciali et al. 2006) is a rich multi-agent framework which supports online reasoning. The agents are built according to the KGP model of agency (Kakas et al. 2004). The authors use an specialized form of the Event Calculus (Kowalski and Sergot 1986) as reasoning formalism. PROSOCS supports planning, reactive behavior, goal revision, plan revision and many more features. Active sensing actions can be specified, but the framework does not support postdiction as part of a contingent planning process: It is not possible to plan for the observation of the effect of an action and then to reason (within the planning) about the condition under which the effect holds. Instead, the framework focuses more on multi-agent aspects.

ExpCog – An Experimental Cognitive Robotics Framework

ExpCog is aimed at integrating logic-based and cognitively-driven agent-control approaches, qualitative models of space and the ability to apply these in the form of planning, explanation and simulation in a wide-range of robotic-control platforms and simulation environments. In addition to its primary experimental function, ExpCog is also geared toward educational purposes. ExpCog provides an easy to use toolkit to integrate qualitative spatial knowledge with formalisms to reason about actions, events, and their effects in order to perform planning and explanation tasks with arbitrary robot platforms and simulators. As demonstrators, support has been included for systems including *ROS*, *Gazebo*, *iCub*. The core integrated agent-control approaches include logic-based approaches like *Situation Calculus*, *Fluent Calculus*, or *STRIPS*, as well as cognitively-driven approaches like *Belief-Desire-Intention*. Furthermore, additional robot platforms and control approaches may be seamlessly integrated. **ExpCog**. (Suchan and Bhatt 2012)

Listing 1 `http://tinyurl.com/expcog`

MAPSIM (Brenner and Nebel 2009) is a continual planning framework based on the planning language MAPL. MAPL is similar to PDDL, but relies on a multi-valued logic. In MAPL, the not-knowing of the value of a certain fluent is modeled with a special `unknown` value. It is not possible to model conditional effects in MAPL, and hence postdiction is not possible. IndiGolog is a high-level programming language by De Giacomo and Levesque (1998) which has a search-operator that can also be used to perform planning. IndiGolog is capable of planning with incomplete knowledge via a generalized search operator (Sardina et al. 2004). However, postdiction and other inference mechanisms have to be implemented by hand, and are thus not elaboration tolerant (McCarthy 1998).

Conclusion and Outlook

We have formalized and implemented an online-planning framework and demonstrated its application in a Smart Home environment. Error-tolerance is achieved by postdicting abnormalities. This requires a formalism like $\mathcal{HPX}$, which supports sensing along with postdiction.

On the application side, work is presently in progress to integrate the online h-approximation of this paper within the general experimental cognitive robotics framework ExpCog (Listing 1; (Suchan and Bhatt 2012)). This integration will make is possible for us to release the online planner in a manner such that it may be seamlessly applied for a wide-range for robot control tasks and existing platforms such as ROS (`http://www.ros.org`). On the theoretical side we are currently investigating domain-independent heuristics and their formalization in terms of ASP to improve the overall performance of the planner: A huge body of research about heuristics in planning can be found in PDDL-planning related literature, but these heuristics are usually formalized and implemented in procedural formalisms. Transferring these ideas to declarative formalisms such as ASP presents many challenges.

References

Baral, C.; Kreinovich, V.; and Trejo, R. 2000. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence* 122(1-2):241–267.

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

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*.

Bhatt, M., and Flanagan, G. 2010. Spatio-temporal abduction for scenario and narrative completion. In Bhatt, M.; Guesgen, H.; and Hazarika, S., eds., *Proceedings of the International Workshop on Spatio-Temporal Dynamics, co-located with ECAI-10*, 31–36. ECAI Workshop Proceedings.

Bhatt, M.; Suchan, J.; and Schultz, C. 2013. Cognitive Interpretation of Everyday Activities — Toward Perceptual Narrative Based Visuo-Spatial Scene Interpretation. In *Computational Models of Narrative (CMN 2013)*, CMN. OASIcs – Open Access Series in Informatics. (to appear).

Bhatt, M. 2012. Reasoning about space, actions and change: A paradigm for applications of spatial reasoning. In *Qualitative Spatial Representation and Reasoning: Trends and Future Directions*. IGI Global, USA.

Bracciali, A.; Demetriou, N.; Endriss, U.; Kakas, A.; Lu, W.; and Stathis, K. 2006. Crafting the Mind of PROSOCS Agents. *Applied Artificial Intelligence*.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems* 19(3):297–331.

De Giacomo, G., and Levesque, H. J. 1998. An incremental interpreter for high-level programs with sensing. Technical report, Department of Computer Science, University of Toronto.

Dubba, K.; Bhatt, M.; Dylla, F.; Hogg, D.; and Cohn, A. 2012. Interleaved inductive-abductive reasoning for learning complex event models. In Muggleton, S.; Tamaddoni-Nezhad, A.; and Lisi, F., eds., *Inductive Logic Programming*, volume 7207 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 113–129.

Eppe, M.; Bhatt, M.; and Dylla, F. 2013. h-approximation: History-Based Approximation to Possible World Semantics as ASP. Technical report, arXiv:1304.4925v1.

Gebser, M.; Grote, T.; Kaminski, R.; and Schaub, T. 2011a. Reactive Answer Set Programming. In *Proceedings of LP-NMR*.

Gebser, M.; Kaminski, R.; König, A.; and Schaub, T. 2011b. Advances in gringo series 3. In *Proceedings of the Eleventh International Conference on Logic Programming and Non-monotonic Reasoning*, number X.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan and Claypool.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Proceedings of the International Conference on Logic Programming (ICLP)*.

Hajishirzi, H., and Mueller, E. T. 2011. Symbolic probabilistic reasoning for narratives. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*. AAAI.

Hajishirzi, H.; Hockenmaier, J.; Mueller, E. T.; and Amir, E. 2012. Reasoning about robocup soccer narratives. *CoRR* abs/1202.3728.

Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS Proceedings*, volume 2005.

Kakas, A.; Mancarella, P.; Sadri, F.; and Stathis, K. 2004. The KGP model of agency. In *ECAI Proceedings*.

Kakas, A.; Kowalski, R.; and Toni, F. 1992. Abductive logic programming. *Journal of logic and computation* 2(6):719.

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

Krieg-Brückner, B.; Röfer, T.; Shi, H.; and Gersdorf, B. 2010. Mobility Assistance in the Bremen Ambient Assisted Living Lab. *GeroPsych: The Journal of Gerontopsychology and Geriatric Psychiatry* 23(2):121–130.

McCarthy, J., and Costello, T. 1998. Combining narratives. In *KR*, 48–59.

McCarthy, J. 1998. Elaboration tolerance. In *Commonsense Reasoning*.

McCarthy, J. 2000. Logic-based artificial intelligence. Norwell, MA, USA: Kluwer Academic Publishers. chapter Concept of logical AI, 37–56.

Miller, R., and Shanahan, M. 1994. Narratives in the situation calculus. *J. Log. Comput.* 4(5):513–530.

Mueller, E. T. 2007. Modelling space and time in narratives about restaurants. *LLC* 22(1):67–84.

Pinto, J. 1998. Occurrences and narratives as constraints in the branching structure of the situation calculus. *J. Log. Comput.* 8(6):777–808.

Poole, D.; Goebel, R.; and Aleliunas, R. 1987. Theorist: A logical reasoning system for defaults and diagnosis. In Cercone, N., and McCalla, G., eds., *The Knowledge Frontier*. Springer. 331–352.

Sardina, S.; Giacomo, G. D.; Lespérance, Y.; and Levesque, H. J. 2004. On the semantics of deliberation in IndiGolog from theory to implementation. *Annals of Mathematics and Artificial Intelligence* 259–299.

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

Suchan, J., and Bhatt, M. 2012. The ExpCog Framework: High-Level Spatial Control and Planning for Cognitive Robotics. In *Bridges between the Methodological and Practical Work of the Robotics and Cognitive Systems Communities - From Sensors to Concepts*. Intelligent Systems Reference Library, Springer. (in press).