Assumption-Based Planning with Sensing via Contingent Planning

Pamela Calvo and **Jorge A. Baier**
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract

Assumption-based planning (ABP) is a recently proposed alternative to conformant planning. Like conformant planning, it is designed for domains in which the initial state is uncertain, and in which actions are deterministic. Unlike conformant planning, an ABP plan may make commonsensical assumptions about the initial state, which may result in simpler, easier-to-communicate plans that achieve the goal provided such assumptions hold.

In this paper we extend the ABP paradigm to domains with sensing. We propose a polynomial compilation of ABP into an extension of conditional planning supporting negation as failure in action preconditions. We show that extending DNF, a well known conditional planner, with negation as failure, is fairly easy. In our theoretical analysis, we prove that our compilation is polynomial, sound, and complete. In our experimental evaluation we use DNF's extension and compare with existing, non-polynomial compilations of ABP to conditional planning, showing that in many domains conditional planners are able to solve instances that could be not solved before.

1 Introduction

Human beings can plan in the presence of high levels of uncertainty with remarkable ease. Arguably, the plans they construct do not always account for all possible contingencies; much to the contrary, they make many assumptions about the state of affairs of the world based on commonsensical criteria. These plans tend to be effective, regardless of how many assumptions have been made. For example, when planning our way home or a trip, we hardly ever build contingency plans for cases in which the city's taxi drivers are on strike and both your car and public transit fail to work altogether.

In contrast, much of the standard AI machinery developed to-date for the problem of planning in the presence of uncertainty has focused on the construction of plans that achieve a goal no matter what is the initial state of the world. While this is desirable in some domains, in many applications one desires a plan that is easy to understand and easy to communicate.

Assumption-based planning (ABP) [Davis-Mendelow *et al.*, 2013] is a recently proposed approach to planning with uncertainty in which a solution plan is composed both by a course of action and a number of assumptions about the states visited during execution. For example, an ABP plan could assume that the subway is operational and return a plan consisting of taking the subway home. Assumptions allow the planning system to explore a wider range of plans, with the potential of returning a simple plan subject to a number of assumptions that are compatible with common sense. ABP may be more applicable than contingent planning in some applications in which uncertainties of the world are not under the control of the planning agent. For example, imagine a situation in which an autonomous rover is unaware of the truth value of a certain fluent f but cannot observe or affect its truth value by performing any action. Imagine further that this prevents the rover from building *any plan* that achieves the goal. In these cases, ABP, unlike other frameworks, provides a way of building a plan which will depend on assuming f or $\neg f$ at a certain state during execution. Whether or not such an assumption is reasonable can be assessed later by a human expert, or achieved by other agents, if feasible.

Davis-Mendelow *et al.*'s approach to ABP is limited to generating plans for unobservable environments. In this paper, we provide a new definition of ABP that incorporates sensing actions. We formalize assumption-based planning with sensing (APBS) as an extension of conformant planning with deterministic actions. We prove that ABPS is not harder than contingent planning: indeed it is 2-EXP-complete. Furthermore, we show how to compile ABPS into a variant of contingent planning in which action preconditions may contain atoms negated under the *negation as failure* semantics. We show that negation as failure in preconditions is a feature that does not make the task of deciding plan existence harder in terms of computational complexity. In addition, it allows us to propose a polynomial-time translation. This property is important and was not enjoyed by Davis-Mendelow *et al.*'s compilation of ABP to classical planning.

We show that implementing negation as failure into the state-of-the-art contingent planner DNF [To *et al.*, 2011] involves the addition of only a handful of lines of code, suggesting that incorporating such a feature to other planners may be just as easy.

In our experimental evaluation, we compare to Davis-

Mendelow *et al.*'s approach in unobservable domains. Results are mixed. While their compilation to classical planning allows exploiting highly optimized classical planners, in some domains their compilation excels possibly because DNF's heuristic is very weak. In other domains, however, Davis-Mendelow *et al.*'s worst-case exponential-time translation either runs out of memory or generates an input that is not handled well by the back-end planner, allowing DNF to solve problems that cannot be solved by Davis-Mendelow *et al.*'s approach.

An approach to finding small plans related to ours is Meuleau and Smith's [2003], which focuses on computing best possible plans with at most k contingencies. As presented, it does not consider the notion of assumptions. Our approach is related but very different from Brafman and Shani's [2012], which implicitly makes assumptions in states based on probability criteria. Our approach does not need probabilities. As motivated above, assumptions do not have to be necessarily facts that are usually true, yet they may be facts that are sometimes *necessary* to find a plan or to find a compact plan, regardless of their likelihood.

2 Preliminaries

The following sections describe the background necessary for the rest of the paper.

2.1 Propositional Logic Preliminaries

Given a set of propositions (or *fluents*) F, the set of *literals* of F, $L(F)$, is defined as $L(F) = F \cup \{\neg p \mid p \in F\}$. A *clause* is a disjunction of literals.

Boolean formulae over F—denoted by $\mathcal{B}(F)$— are defined inductively as usual, and may contain constants $\top$ and $\bot$, which are used to denote "true"and "false", and standard connectives ($\wedge$, $\neg$). We assume the $\models$ relation to be defined in the standard way; that is, a formula φ is entailed by a set of clauses $\mathcal{C}$, denoted by $\mathcal{C} \models \varphi$, iff all models of $\mathcal{C}$ satisfy φ. If F is a set of propositions, we define the set of literals with negation as failure as $L_{not}\ (F) = L(F) \cup \{not\ p \mid p \in L(F)\}$. If $p \in F$, and $\mathcal{C}$ is a set of clauses, we say that $\mathcal{C} \models not\ p$ iff $\mathcal{C} \not\models p$. Finally, we say $\mathcal{C} \models L$ if $\mathcal{C} \models \ell$, for every $\ell \in L$.

A set of literals over F is said to be *logically complete* if it contains either p or $\neg p$, for every $p \in F$. If $p \in F$, then $\neg p$ is the complement of p and p is the complement of $\neg p$. If ℓ is a literal, we denote its complement by $\bar{\ell}$. A set of literals over F is *consistent* if it does not contain a pair of complementary literals. If S is a set of sets of literals, then we say that $S \models L$, iff for every $s \in S$, it holds that $s \models L$.

2.2 Contingent and Assumption-Based Plans

Below we follow closely the definitions of Davis-Mendelow *et al.* [2013] for conformant plan and assumption-based plan.

Definition 1 (Planning Problem) *A planning problem is a tuple $P = (F, O, I, G)$ where F is a finite set of fluents, O is a finite set of actions, I is a set of clauses over F, defining the set of possible initial states, and G is a boolean formula over symbols in F, that defines a goal condition.*

For every action a, function $prec(a)$, the *precondition* of a, returns a subset of $L(F)$. Furthermore, $\mathit{eff}(a)$, the *effect* of a, is a set of contingent effects, each of the form $C \rightarrow \ell$, where $C \subseteq L(F)$ and $\ell \in L(F)$.

Example Imagine a situation in which we want to obtain a plan to go from home to the office. There are three ways of getting there: walking, by bus, or by subway. The actions are $bus(A, B)$, which takes a bus from A to B, $subway(A, B)$, which takes the subway from A to B, and $walk(A, B)$ which, by foot, gets you from A to B. Action $bus(A, B)$ is executable when there is no driver strike (denoted by fluent $strike$). Action $walk(A, B)$ can be performed if the agent is at A. Formally, we define these actions as follows. For every $x \in \{home, subwayH\}$:

$$prec(bus(x, \mathit{office})) = \{\neg strike, at(x)\}$$
$$\mathit{eff}(bus(x, \mathit{office})) = \{\neg at(x), at(\mathit{office})\}$$

For every $x, y \in \{home, \mathit{office}, subwayH, subwayO\}$ such that $x \neq y$, we define

$$prec(walk(x, y)) = \{at(x)\}$$
$$\mathit{eff}(walk(x, y)) = \{\neg at(x), at(y)\}$$

For every $x, y \in \{subwayH, subwayO\}$, where $x \neq y$, we define

$$prec(subway(x, y)) = \{operational, at(x)\}$$
$$\mathit{eff}(subway(x, y)) = \{\neg at(x), at(y)\}$$

♦

A *planning state* for P is a set of literals over F that is both logically complete and consistent. An action a is *applicable* in a planning state s iff $s \models prec(a)$. We denote by $\delta(s, a)$ the state that results from applying a in s. Formally,

$$\delta(s, a) = (s \setminus \{\bar{\ell} \mid C \rightarrow \ell \in \mathit{eff}(a), s \models C\}) \cup \{\ell \mid C \rightarrow \ell \in \mathit{eff}(a), s \models C\}$$

if $s \in F$ and a is applicable in s; otherwise, $\delta(s, a)$ is undefined. It is convenient to extend the definition of δ for sequences of actions. If α is a sequence of actions and a is an action, we define $\delta(s, \alpha a)$ as $\delta(\delta(s, \alpha), a)$ if $\delta(s, \alpha)$ is defined. Furthermore, if α is the empty sequence, then $\delta(s, \alpha) = s$.

Definition 2 (Execution Trace) *Given a planning problem $P = (F, O, I, G)$ and a sequence of actions $\alpha = a_0 a_1 \ldots a_n$, we say that α induces an execution trace $\sigma = s_0 s_1 \cdots s_k$ iff*

1. *$I \models s_0$, and s_0 is a planning state.*
2. *$\delta(s_i, a_i) = s_{i+1}$, for all $i < k$, and*
3. *either $k = n+1$ or $k < n+1$ and $\delta(s_k, a_k)$ is undefined.*

Definition 3 (Successful Execution Trace) *An execution trace σ for α is successful iff $|\sigma| = |\alpha| + 1$.*

Definition 4 (Leads to) *An execution trace $\sigma = s_0 \cdots s_k$ leads to (goal formula) G, iff $s_k \models G$.*

Definition 5 (Conformant Plan) *A sequence of actions α is a conformant plan for $P = (F, O, I, G)$ iff every execution trace of α is successful and leads to G.*

Definition 6 (Conforms to) *An execution trace $\sigma = s_1 \cdots s_k$ conforms to a sequence of boolean formulae $\rho = h_1 \cdots h_n$ with $k \leq n$ iff $s_i \models h_i$, for every $i \in \{1, \ldots, k\}$.*

Finally, each of the execution traces of α that conform to ρ must actually lead to the goal. A formal definition of an assumption-based plan follows.

Definition 7 (Assumption-Based Plan) *The pair (ρ, α), where α is a sequence of k actions, and ρ is a sequence of $k+1$ boolean formulae over T is an assumption-based plan for $P = (F, O, I, G, T)$ iff any execution trace of α that conforms to ρ is successful and leads to G, and furthermore at least one such execution trace exists.*

Example (continued) Assume we define the initial state as $I = \{at(home)\}$, and the goal as $G = at(office)$. Note this means that it is not known whether the subway is operational or whether their is a bus strike. Then the only conformant plan for the problem is given by $walk(home, office)$.

However, if $operational$ is an assumable fluent, then (ρ_1, α_1) is an ABP plan, with:

$$\begin{aligned} \rho_1 &= operational, \top, \top, \top \\ \alpha_1 &= walk(home, subwayH), \\ &\quad subway(subwayH, subwayO), \\ &\quad walk(subwayO, office) \end{aligned}$$

Finally, if $strike$ is assumable, then (ρ_2, α_2) is also an ABP plan, with:

$$\begin{aligned} \rho_2 &= \neg strike, \top \\ \alpha_2 &= bus(home, office) \end{aligned}$$

♦

2.3 Contingent Planning

Contingent planning extends conformant planning by allowing the agent to observe the world via sensing actions. As such, we assume the set of action operators is partitioned into two (disjoint) sets O_w and O_s, which contain, respectively, the actions that modify the world and sensing actions.

Formally a contingent planning problem is characterized by a tuple (F, O_w, O_s, I, G), where F, I, and G are defined as above. Likewise, functions $prec$ and eff return the precondition and effect of each action a in O_w. For every sensing action $o \in O_s$, we associate a precondition $prec(o)$ specifying the conditions under which o is executable, and we associate an observation $obs(o)$, which is a Boolean formula over F specifying the condition that is sensed by o.

In general, contingent plans look like programs with if-then-else conditions. In the formalization, we focus on tree-like programs to simplify our definitions. Tree-like programs are as expressive as programs with if-then-else constructs.

Definition 8 (Contingent Program) *A contingent program for contingent planning task (F, O_w, O_s, I, G) is the language for variable* prog *of the following BNF grammar.*

$$\begin{aligned} \mathsf{act} &::= a && \text{(for every } a \in O_w\text{)} \\ \mathsf{obs} &::= o && \text{(for every } o \in O_s\text{)} \\ \mathsf{prog} &::= \varepsilon && \text{(the empty program)} \\ \mathsf{prog} &::= \mathsf{act} \cdot \mathsf{prog} && \text{(for } a \in O_w\text{)} \\ \mathsf{prog} &::= branch(\mathsf{obs}, \mathsf{prog}, \mathsf{prog}) && \text{(for } o \in O_s\text{)} \end{aligned}$$

Just like any action sequence induces an execution trace of states, so do contingent plans. Providing a formal definition for these traces will ultimately allow us to give a formal definition for a plan. First, however, we need an intermediate step: we need to define what does it mean to execute a program. We do this by first defining the notion of configuration.

Given a contingent planning problem, a configuration is a pair (S, p), where S is a set of planning states—also referred to below as *belief state*—and p is a program. Now we define the $\rightsquigarrow$ relation which can be intuitively related to atomic execution steps. More precisely, if $(S, p) \rightsquigarrow (S', p')$ then the execution of one atomic step of p in S leads to state S', with p' remaining to be executed. Formally,

1. $(S, a \cdot p) \rightsquigarrow (S', p)$, for every $a \in O_w$, if and only if $S \models prec(a)$ and $S' = \{\delta(s, a) \mid s \in S\}$.
2. $(S, branch(o, p_1, p_2)) \rightsquigarrow (S', p)$, for every $o \in O_s$, if and only if $S \models prec(o)$, and either $S' = T$ and $p = p_1$ or $S' = S \setminus T$ and $p = p_2$, where $T = \{s \in S \mid s \models obs(o)\}$.

Now we formally define a trace of execution for a program.

Definition 9 (Trace of Configurations) *Given a contingent planning problem (F, O_w, O_s, I, G), and a program p, we say p induces an execution trace $c_0 c_1 \ldots c_n$ iff:*

1. *$c_0 = (S_0, p)$, where $S_0 = \{s \mid s$ is a state and $I \models s\}$.*
2. *$c_{i-1} \rightsquigarrow c_i$, for every $i \in \{1, \ldots, n\}$.*
3. *there is no c such that $c_n \rightsquigarrow c$.*

As before successful traces are those that involved executing every single action of the program.

Definition 10 (Successful Trace of Configurations) *A trace of configurations $c_1 \ldots c_n$ is successful if and only if $c_n = (S, \varepsilon)$, for some s.*

Definition 11 (Leads to, for Configurations) *A trace of configurations $c_0 c_1 \ldots c_n$ leads to G if $c_n = (S, p)$ and $S \models G$.*

Now we are ready to define contingent plans.

Definition 12 (Contingent Plan) *A contingent program p is a plan for $P = (F, O_w, O_s, I, G)$ if and only if every trace of configurations of p over P is successful and leads to G.*

Example (continued) Assume the initial state and goal state are defined as above, and that in addition we have the following sensing actions to sense whether or not the subway is operational. For each $x \in \{stnO, stnH\}$:

$$\begin{aligned} prec(senseOp(x)) &= \{at(x)\} \\ obs(senseOp(x)) &= operational \end{aligned}$$

In addition to the plan $walk(home, office)$, we have the contingent plan:

$$walk(home, stnH) \cdot branch(senseOp(stnH), p_1, p_2)$$

where

$$p_1 = subway(stnH, stnO) \cdot walk(stnO, office),$$

and where p_2 may describe various plans that involve waking. Important is the fact that p_2 may not consider taking the bus, since fluent $strike$ is unknown, unobservable, and its truth value cannot be changed by the agent. ♦

3 Negation as Failure in Preconditions

Our translation uses a version of contingent planning that requires negation as failure (NAF) in action preconditions. This is not standard in contingent planning systems [Hoffmann and Brafman, 2005; Albore *et al.*, 2009; To *et al.*, 2011]. Nevertheless, as we show in this section, it is easy to extend a state-of-the-art contingent planner with NAF in preconditions. In the rest of the section, we first show that adding support for NAF in preconditions does not change the complexity class of contingent planning. Then, we show how to incorporate this feature to a state-of-the-art planner.

3.1 A Note on Complexity

NAF in preconditions is related to modal logic modalities in preconditions [Bonet, 2010]. Bonet proved that adding modal operators like $\Box p$ ("always p") and $\Diamond p$ ("possibly p") to preconditions does not change the complexity class of contingent planning, for the more general case of finding an unconstrained branching plan (Theorem 2, Bonet 2010). We can do likewise for the case of NAF.

In short, our proof follows from three facts. First, plan existence for a contingent planning problem is a 2-EXP-complete decision problem [Rintanen, 2012]. Second, 2-EXP is equivalent to AEXPSPACE; that is, the class of problems that can be decided with an alternating Turing machine (ATM) that uses exponential space. This means there exists an exponential-space ATM, say, M, that decides existence of a contingent plan. Third, we can modify M to now decide existence of a plan for a contingent problem with NAF preconditions, without requiring more memory. Indeed, such a machine is a slight modification of M. The only modification is the module in which preconditions are checked. For contingent planning, for every $\ell \in prec(a)$ we need to check $S \models \ell$, where the size of S may be exponential in the size of the problem ($|S|$ is worst-case the set of all possible states). Note that $S \models \ell$ does not require more than exponential space since it can be done in non-deterministic polynomial time in $|S| + |\ell|$. Adding support for NAF requires checking $S \not\models \ell$, which clearly can also be done in exponential space in the size of the problem. The rest of the M is left with no further modifications. Formally,

Theorem 1 *Contingent planning with NAF in preconditions is in 2-EXP.*

Corollary 1 *Contingent planning with NAF in preconditions is 2-EXP-complete.*

Proof: Follows from the fact that contingent planning, a 2-EXP-complete problem is a particular case of contingent planning with negation as failure. ■

3.2 Implementing Negation as Failure in DNF

Now we turn into a more practical aspect of negation as failure: its implementation in a state-of-the-art system, namely, DNF [To *et al.*, 2011].

As its name suggests, DNF represents belief states using disjunctive normal form; that is, as disjunctions of literal conjunctions. Internally, belief states are represented as sets of conjunctions. Algorithm 1 shows a pseudocode for the C++ routine actually used in DNF to check the applicability of an action. Essentially, it checks that every literal in $prec(a)$ is in every conjunction c of the belief state cs. Extending Algorithm 1 to support negation as failure is extremely easy. Algorithm 2 shows how we did it in our implementation. The main difference is in Lines 2–8, which will declare that $cs \models not\ \ell$ if cs contains a conjunction in which ℓ does not appear.

Algorithm 1: DNF's precondition evaluation

```
   Input: An action a, a set of conjunctions of literals cs
1  for each conjunction c in cs do
2      for each ℓ ∈ prec(a) do
3          if ℓ ∉ c then
4              return false
5  return true
```

Algorithm 2: Precondition evaluation with NAF

```
   Input: An action a, a set of conjunctions of literals cs
1  exec ← true
2  for each "not ℓ" in prec(a) do
3      for each conjunction c in cs do
4          if ℓ ∉ c then
5              exec ← true
6              break
7      if not exec then
8          return false
9  for each conjunction c in cs do
10     for each ℓ ∈ prec(a) ∩ L(F) do
11         if ℓ ∉ c then
12             return false
13 return true
```

4 Assumption-based Planning with Sensing

Now we extend the definition of ABP to the case of sensing. At the formal level an ABPS instance is simply a tuple (F, O_w, O_s, I, G, T), where all elements are defined as in conformant planning, and $T \subseteq F$ are the assumable fluents.

As with regular ABP, we want to allow an assumption prior to each action execution. Plans for an ABPS problem look like contingent programs where each world action or sense action is accompanied with an assumption. We call these programs assumption-based contingent programs.

Definition 13 (Assumption-based Contingent Program) *An assumption-based contingent program is any element of the language defined by the* prog *variable of the BNF grammar of Definition 8, when replacing the rules for* act *and* obs *by the following:*

$$\mathsf{act} ::= [h, a] \qquad \text{(for every } a \in O_w, h \in \mathcal{B}(F))$$
$$\mathsf{obs} ::= [h, o] \qquad \text{(for every } o \in O_s, h \in \mathcal{B}(F))$$

Now we adapt the definitions for contingent planning to consider the effect of an assumption. The main difference between ABPS and contingent planning is that assumptions *filter out* some of the states in belief states; specifically, those states that are inconsistent with the assumption. We only require the goal to hold in the worlds that have not been filtered out. An additional requirement is that assumptions cannot filter out all states of the belief state. This is needed for two reasons. First, we do not want to allow assumptions that are inconsistent with all states in the belief state. Second, by definition, every action is executable in an empty belief state, and every formula is satisfiable in an empty belief state. As such, if we don't prevent empty belief states to arise, we would obtain plans with an arbitrary number of spurious actions.

We formalize the notion of filter with the following definition $\mathrm{Filter}(S, h) = \{s \in S \mid s \models h\}$. Relation $\rightsquigarrow$ for ABPS is defined in the following way.

1. $(S, [h, a] \cdot p) \rightsquigarrow (S', p)$, for every $a \in O_w$, if and only if $S_h = \mathrm{Filter}(S, h)$ is nonempty, $S_h \models prec(a)$ and $S' = \{\delta(s, a) \mid s \in S_h\}$.
2. $(S, branch([h, o], p_1, p_2)) \rightsquigarrow (S', p)$, for every $o \in O_s$, if and only if $S_h = \mathrm{Filter}(S, h)$ is nonempty, $S_h \models prec(o)$, and either $S' = T$ and $p = p_1$ or $S' = S \setminus T$ and $p = p_2$, where $T = \{s \in S_h \mid s \models obs(o)\}$.

Definition 14 (Assumption-based Plan with Sensing) *A assumption-based contingent program p is a plan for $P = (F, O_w, O_s, I, G, T)$ if and only if every trace of configurations of p over P is successful and leads to G.*

Example (continued) Suppose, like above, that $operational$ can be sensed by action $senseOp$, but that $strike$ can be assumed but cannot be sensed by any action. Then the following is an ABPS plan for the resulting problem.

$$[\top, walk(home, stnH)] \cdot [\top, branch(senseOp(stnH), p_1, p_2)]$$

where

$$p_1 = [\top, subway(stnH, stnO)] \cdot [\top, walk(stnO, office)],$$

and where p_2 can now consider an action that takes the bus, while assuming there is no strike. Indeed, p_2 may be $[\neg strike, bus(stnH, office)]$. ♦

Note that our definition does not need to define a notion of conformity (cf. Definition 7), because we ensure a plan conforms to the assumptions by applying the Filter function. In the absence of sensing actions, Definitions 14 and 6 are equivalent, for those plans that do not assume anything about the final state.

Theorem 2 *Let $P = (F, O, I, G, T)$ and $P' = (F, O, \emptyset, I, G, T)$ be respectively ABP and ABPS problems. Then $(h_0 \dots h_n \top, a_0 \dots a_n)$ is a plan for P if and only if $[h_0, a_0] \cdots [h_n, a_n]$ is a plan for P'.*

Finally, ABPS is in the same complexity class as contingent planning.

Theorem 3 *Deciding existence of an ABPS plan is 2-EXP-complete.*

Proof: Follows from the correctness of the translation we propose in the following section. ■

5 ABPS via Contingent Planning

Our translation takes an ABPS problem $P = (F, O_w, O_s, I, G, T)$ as input and produces a contingent planning problem with NAF preconditions, $P' = (F', O'_w, O'_s, I', G)$. The assumable fluents in T are translated into a set of world actions and sensing actions that, by being applied in a certain order, will have the same effect of making an assumption about the current state of the world. Notice that G is the only set that remains unmodified.

To understand the intuition underlying our translation, we first observe that assumptions can be related to sensing actions. Indeed, a sensing action o will, in one branch, filter out from the belief those states inconsistent with $obs(o)$ whereas in the other branch those states consistent with the observation are filtered out. For sensing actions a plan is needed for both branches. For assumptions, on the other hand, we are only interested in the belief state that has filtered out those states inconsistent with the assumption.

One way of mapping an assumption into a sensing action would be to have a sensing action capable of performing both an observation and a change in the world. If t is an assumable fluent, one would create a sensing action whose observation is t. In addition, we would add a conditional effect $\neg t \rightarrow g$, for every g in the goal G.[1] Such a sensing action would immediately "solve the problem" in the branch that is inconsistent with t. Unfortunately, the current model for contingent planning that planning system support does not handle conditional effects in sensing actions.

What our compilation does is to model assumptions with 4 distinguished actions for each assumable fluent. First, and foremost, there is a sensing action whose observation is the assumable fluent; its role is to filter out states that are consistent/inconsistent with the assumption. Second, there is a world action ($assumme(t)$) that intuitively starts "assumption mode". An effect of this action is the fluent $lock$, which does not allow other world actions to be performed as soon as it becomes true. Then there are actions $done$ and $unlock$. Both of them can only be performed after the sensing action. Action $unlock$, as the name implies, makes $lock$ false, while action $done$, makes the goal true.

The details of the compilation are as follows.

[1] We assume here the goal is a set of literals to simplify the presentation, but the compilation can be adapted if G is a Boolean formula.

Fluents The set of fluents of the compiled problem is defined as:

$$F' = F \cup \{lock\} \cup F_A,$$

where fluent $lock$ resembles an "assumption mode" in which regular actions from the original problem (those in O_w or O_s) are not executable. When $lock$ is true, only assumption-related actions are applicable. Consequently, when it is false, only actions that belong exclusively to O_w or O_s are applicable. Finally, $F_A = \{assuming(t) \mid t \in T\}$. Fluent $assuming(t)$ intuitively indicates that "t is being assumed". Its dynamic should be clear after reading how actions are defined.

Actions All actions in P are "copied into" P', but their precondition is modified to include $lock$. Formally,

$$O'_w = \{\hat{a} \mid a \in O_w\} \cup A_w$$
$$O'_s = \{\hat{a} \mid a \in O_s\} \cup A_s$$

We modify the preconditions of actions in $O_w \cup O_s$ to include the $lock$ fluent. Formally,

$$prec(\hat{a}) = prec(a) \cup \{\neg lock\}, \text{ for each } a \in O_w \cup O_s$$

Below we define A_w and A_s constructively, assuming they are initially empty. For each $t \in T$ that assumes a literal p to be true, we add:

1. An action $assume(t) \in A_w$. This action initiates the process of making an assumption.

   $$prec(assume(t)) = \{\neg lock, not\ t, not\ \neg t\}$$
   $$\mathit{eff}(assume(t)) = \{assuming(t), lock\}$$

   Since an assumption will be reduced to several world and sensing actions applied in an specific order, the fluent $assuming(t)$ is needed to ensure the correct course of action execution. Note that this is the action that requires NAF in preconditions. Essentially the precondition is saying that we can only assume a fluent t if neither t nor $\neg t$ is true; in other words, we can assume t if t is unknown.

2. A sensing action $assume\text{-}s(t) \in A_s$. This is the next action to be applied. It will split the current belief state in two: one containing states where t is true and one containing the remaining states, where $\neg t$ is true. When making an assumption only the branch where t is true is relevant, and so two more actions are needed in order to finish the process of making an assumption. This will be explained in the following two steps. Before that, we specify the preconditions for this action.

   $$prec(assume\text{-}s(t)) = \{assuming(t), lock\}$$

   This action does not have any effects besides the branching due to its sensing nature.

3. An action $done \in A_w$. The previously described sensing action will branch the plan into two paths. Since we are making an assumption, there is one branch which is not needed anymore and therefore has to be pruned so the plan does not continue expanding it. The action $done$ is the one responsible for finishing off this branch. It does this by adding all goal fluents.

   $$prec(done(t)) = \{lock, \neg t, assuming(t)\}$$
   $$\mathit{eff}(done(t)) = \{g \mid g \in G\}$$

4. Finally, $unlock(t)$ removes the $lock$ fluent to allow the planner to continue using world and sensing actions (by removing $lock$) in the branch that is consistent with the assumption (i.e., in which t holds). Its definition follows.

   $$prec(unlock(t)) = \{lock, t, assuming(t)\}$$
   $$\mathit{eff}(unlock(t)) = \{\neg lock, \neg assuming(t)\}$$

Initial State The new initial state of the problem is:

$$I' = I \cup \{\neg lock\} \cup \{\neg f \mid f \in F_a\}.$$

6 Empirical Evaluation

A note on DNF's performance In earlier stages of the experimental phase, we noticed that DNF was prone to choose actions that led to just one belief state as a result, over than those that generated a new one (like sensing and assumption actions). This was because the PrAO* algorithm [To *et al.*, 2011] attempts to minimize the branching in the returned plans, which is not good if we want to generate plans with assumptions. To minimize this effect, we modified the heuristic modified the heuristic to increase likelihood of choosing nodes that come from a branching action. We called this variant DNF-ABPS^A.

We designed two sets of experiments. In the first, we compared with our ABP predecessor, A0+Lama [Davis-Mendelow *et al.*, 2013] in problems that do not have sensing actions. We evalueated over two of the four domains that were evaluated by Davis-Mendelow *et al.* [2013]: *alogistics* a modified version of *logistics* in which assumptions are needed to get to the goal, and the well-known *coins* domain, which included assumption actions that allow assuming the initial location of coins. Results are shown Table 1.

In our second set of experiments, the objective was to compare a contingent planner with an ABPS planner on the same problem, which the conformant planner does not, but the intent here was to evaluate difference on performance and on plan size. Of course, the ABP planner has access to a set of assumption actions. We ran DNF (without assumptions), and the two versions of DNF that used our translation (DNF-ABPS and DNF-ABPS^A) over modified conformant domains *dispose* and *push-to*. In these domains we are can assume the position of the objects. Finally, we created *medical-allergy* inspired by IPC's *medical*. In this domain there is a fixed number of illnesses and medicines that can cure each. Unfortunately, treatments may cause an allergic reaction, not deadly, but undesired. To determine whether or not a person is allergic, a number of actions that have to be performed. Furthermore, if the person turns out to be allergic to a medicine, there is an alternative medicine available, but it has to be imported (this was simulated by a number of actions). In this problem one can assume that a patient is not allergic. Results for all three domains are shown in Table 2.

	A0 + Lama			DNF-ABPS					DNF-ABPS^A			
Problem	Total T	Sol Len	#Ass	Trans T	Plan T	Total T	Sol Len	#Ass	Plan T	Total T	Sol Len	#Ass
alog-01	445,31	**35**	2	0,278	32,526	**32,804**	112	2	131,230	131.508	109	2
alog-02	**0,02**	**36**	3	1,145	TO	-	-	-	TO	-	-	-
alog-04	**0,03**	**36**	3	83,753s	TO	-	-	-	TO	-	-	-
alog-06	**1,45**	**41**	4	0,374	TO	-	-	-	1470,01	1470,384	568	5
alog-10	**0,02**	**33**	4	0,294	1739,744	1740,038	132	5	21,989	22,283	151	5
Coins 01	**0,01**	**5**	3	0,166	0,014	0,18	18	2	0,019	0,185	16	2
Coins 07	**0,08**	**8**	5	0,198	1,038	1,236	36	4	1,356	1,554	30	4
Coins 13	**4,29**	**10**	7	0,316	TO	-	-	-	TO	-	-	-
Coins 19	**214,96**	**7**	7	0,359	TO	-	-	-	TO	-	-	-
Coins 25	**129,12**	**16**	16	9,582	TO	-	-	-	TO	-	-	-
Coins 30	**398,32**	**52**	21	17,45	TO	-	-	-	TO	-	-	-

Table 1: T0 + Lama, DNF-ABPS y DNF-ABPS^A, *alogistics* and *coins*. Times in seconds.

	DNF-ABPS					DNF-ABPS^A				DNF-Cont			
Problem	Trans T	Plan T	Total T	Sol Len	#Ass	Plan T	Total T	Sol Len	#Ass	Trans T	Plan T	Total T	Sol Len
push-to-4-3	0,459	51,113	51,572	108	2	18,695	19,154	**50**	3	0,457	14,987	**15,354**	107
push-to-5-3	1,427	742,415	743,482	135	0	159,063	160,49	**44**	3	1,622	106,629	**108.251**	213
push-to-5-4	2,174	TO	-	-	-	TO	-	-	-	2,386	TO	-	-
push-to-5-5	3,127	TO	-	-	-	TO	-	-	-	3,57	TO	-	-
push-to-8-1	9,396	2,388	11,784	42	1	0,715	**10,111**	**13**	1	10,912	0,976	11.888	135
push-to-8-2	23,258	423,714	446,972	154	1	78,639	**101,897**	**49**	2	27,269	81,760	109,029	232
push-to-8-3	49,459	TO	-	-	-	TO	-	-	-	52,007	TO	-	-
push-to-8-10	516,833	TO	-	-	-	TO	-	-	-	8m37.427	TO	-	-
push-to-10-1	63,729	11,234	74,963	**69**	1	7,344	71,073	88	1	63,404	3.227	**66,631**	326
push-to-10-2	147,608	TO	-	-	-	456,567	604,175	**67**	2	154,947	414.303	**569,25**	308
dispose-4-5	0,35	TO	-	-	-	TO	-	-	-	0,282	TO	-	-
dispose-7-3	2,41	TO	-	-	-	TO	-	-	-	2,51	TO	-	-
dispose-7-6	3,812	TO	-	-	-	TO	-	-	-	3,331	TO	-	-
dispose-10-1	28,765	1,427	**30,192**	**16**	1	2,22	30,985	34	1	28,903	1,853	30,756	400
dispose-10-4	32,47	TO	-	-	-	TO	-	-	-	39,804	TO	-	-
medical001	0,161	0,006	0,167	**5**	1	0,004	**0,165**	**5**	1	0,144	0,027	0,171	11
medical002	0,149	0,193	0,342	28	1	0,159	0,308	**19**	2	0,176	0,048	**0,224**	45
medical003	0,145	0,229	0,374	55	2	0,231	0,376	**27**	3	0,158	0,131	**0,289**	113
medical004	0,162	0,548	0,71	67	3 (5)	0,545	0,707	**34**	4	0,16	0,445	**0,605**	300
medical005	0,157	1,856	2,013	104	4(7)	0,365	**0,522**	**53**	5	0,155	1,191	1,346	620

Table 2: DNF-ABPS, DNF-ABPS^A, and DNF-Contingent, *push-to*, *dispose*, and *medical-allergy*. Times in seconds.

Experimental Conclusions A0, which uses LAMA [Richter *et al.*, 2008] as a back-end planner, performs significantly better in the *alogistics* domain. This is possibly due to the fact that the DNF uses a very weak heuristic (goal counting), and does not exploit the advanced search techniques (like preferred operators) that are key to LAMA's performance [Richter and Helmert, 2009].

In the *push-to* domain, on the other hand, A0 cannot translate almost any instance, and DNF and our approach can solve many of them in a couple of seconds or minutes, depending on the complexity of the problem. While observing the result for the different versions of DNF over the *push-to* and *dispose* domains, we can note, firstly that the translation time is not significantly different when assumption actions are added. Secondly, it is easy to see that the modification over DNF-ABPS improves its performance and balances the priority of world actions and branching actions. Hence, more assumptions are made in plans found by DNF-ABPS^A. Thirdly, observing the characteristics of the plans themselves, DNF tends to take a shorter time to give a solution but these solutions are the longest. DNF-ABPS obviously takes more time to find a solution since its search space is wider and since it penalizes assumption actions, it doesn't have a quick escape route to simplify it. In contrast, DNF-ABPS^A, by doing more assumptions, provides considerably shorter solutions.

Finally, in *medical-allergy*, when the planner has more available assumptions, planning takes longer (probably due to the increasing branching factor), but solutions are shorter. Most likely, best results in both aspects, time and length, would come from a domain that allows assumptions and yet puts some kind of restrictions over what assumption can be performed. In future work, we would like effect on performance and plan quality when varying the set of assumable fluents in various ways.

7 Summary and Future Work

We proposed an extension of ABP for domains in which sensing actions are available. We proposed a polynomial-time translation of ABP problems with sensing to contingent planning with negation as failure. Even though negation as failure is not a standard feature of contingent planners, we show that modifying one such a planner (DNF) is easy to do.

In our evaluation, we confirm that using assumptions may reduce plan size. In comparison to A0, the translator of the previous approach to ABP that does not support sensing and uses classical planners as a back end, we observe that in some cases the optimized classical planner outperforms the contingent planner we use. In other cases, however, our polynomial-time translation seems to pay off, allowing our planner to solve instances that cannot be translated by A0.

In future work we plan to extend our experimental evaluation to more domains, and will evaluate the effect of using different sets of assumable fluents. We also will investigate different quality objectives (e.g., smaller plans, fewer assumptions). Another line of future research is the integration of these types of assumptions into logic-programming planning frameworks (e.g., Tu *et al.* 2007).

References

[Albore *et al.*, 2009] Alexandre Albore, Héctor Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1623–1628, Pasadena, CA, 2009.

[Bonet, 2010] Blai Bonet. Conformant plans and beyond: Principles and complexity. *Artificial Intelligence*, 174(3-4):245–269, 2010.

[Brafman and Shani, 2012] Ronen I. Brafman and Guy Shani. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research*, 45:565–600, 2012.

[Davis-Mendelow *et al.*, 2013] Sammy Davis-Mendelow, Jorge A. Baier, and Sheila A. McIlraith. Assumption-based planning: Generating plans and explanations under incomplete knowledge. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

[Hoffmann and Brafman, 2005] Jörg Hoffmann and Ronen Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 71–80, Monterey, CA, USA, June 2005. Morgan Kaufmann.

[Meuleau and Smith, 2003] Nicolas Meuleau and David E. Smith. Optimal limited contingency planning. In *Proceedings of the Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 417–426, Acapulco, Mexico, August 2003.

[Richter and Helmert, 2009] Silvia Richter and Malte Helmert. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

[Richter *et al.*, 2008] Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 975–982, Chicago, IL, 2008.

[Rintanen, 2012] Jussi Rintanen. Complexity of conditional planning under partial observability and infinite executions. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, pages 678–683, 2012.

[To *et al.*, 2011] Son Thanh To, Enrico Pontelli, and Tran Cao Son. On the effectiveness of CNF and DNF representations in contingent planning. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2033–2038, 2011.

[Tu *et al.*, 2007] Phan Huy Tu, Tran Cao Son, and Chitta Baral. Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. *Theory and Practice of Logic Programming*, 7(4):377–450, 2007.