Planning as abduction

Andreas Herzig **Jérôme Lang**
IRIT
Université Paul Sabatier
31062 Toulouse Cedex (France)
{herzig, lang}@irit.fr

Pierre Marquis
CRIL
Université d'Artois
62307 Lens Cedex (France)
marquis@cril.univ-artois.fr

Abstract

We show that while planning in deterministic domains with a fixed horizon can be encoded as a satisfiability problem, assuming that actions are nondeterministic leads to a more complex problem, whose complexity (in the general case) makes a polynomial translation into a satisfiability problem impossible (under the standard assumptions of complexity theory). We show that a logical operation which suits well to planning under incomplete knowledge is abduction. We study the problem successively in the cases where the environment is unobservable and fully observable.

1 Introduction

For a few years, there has been a considerable interest in the planning community towards the "planning as satisfiability" paradigm, initiated by the seminal paper by Kautz and Selman [Kautz and Selman, 1992]. The expression of a planning problem into a satisfiability problem consists in (i) fixing the horizon to a given constant N, (ii) associating to each (action or state) variable v and each time point $t \in \{0, \ldots, N\}$ a new variable v_t; (iii) expressing the dynamics laws, i.e., the deterministic effects of actions, for each action and each time point, within this extended language; (iv) expressing that for each time point, exactly one action variable is true; (v) expressing the goal as a formula labelled by the last time point N. The key result is that any model of the formula consisting of the conjunction of the action laws and the goal, when restricted to the action variables, is a valid plan, and if no model exists, then no valid plan exists. Accordingly, searching for plans is reduced to searching for models of propositional formulas, which can be achieved thanks to many algorithms, including "efficient" (but incomplete) local search procedures.

The fact that it is possible to express a planning problem as a satisfiability problem using a polynomial translation implies that the plan existence problem tackled is in NP. Conversely, we know that if plan existence is in NP, then there necessarily exists a polynomial reduction of it into a satisfiability search (which does not mean that this reduction is useful in practice). Determining the range of planning as satisfiability amounts therefore to finding classes of planning problems which are in NP. The following easy result slightly generalizes the membership results by Bylander [Bylander, 1994] and Baral et al. [Baral *et al.*, 1999].

Proposition 1 *Consider a propositional action description language $\mathcal{L}_A$ (each action a is described in $\mathcal{L}_A$ by a formula $Desc(a)$). Suppose that $\mathcal{L}_A$ verifies the following property: for each* deterministic *action a, the resulting state $Res(s, a)$ of action a in state s is computable in polynomial time from s and $Desc(a)$. Then the existence of a plan of polynomially-bounded length for an initial state s_0, a set A of deterministic actions and a goal G (expressed as a propositional formula) is in* NP.

The assumption that the result of a deterministic action is computable in polynomial time is verified at least by two action languages, namely, STRIPS and $\mathcal{A}$, which leads to the complexity results for polynomial-bound planning in deterministic environments established in [Bylander, 1994] and [Baral *et al.*, 1999].

Without fixing the language $\mathcal{L}_A$ it is difficult to express a completeness result. However, it is clear that NP-completeness is easily reached. In particular, it is reached for $\mathcal{L}_A$ being STRIPS [Bylander, 1994] and for $\mathcal{L}_A$ being $\mathcal{A}$ [Baral *et al.*, 1999]. More generally, polynomial-length plan existence under the assumptions of Proposition 1 is NP-completa. When the latter assumption is not true, polynomial plan existence can be decided in polynomial time by requiring some syntactical restrictions on the effects of actions [Bylander, 1994].

It is important to note that each of the required assumptions (deterministic actions, initial state given, plans of polynomial size and polynomially computable results) is necessary to guarantee membership to NP. The aim of this paper is to see what is happening when relaxing the determinism assumption (and, in the same spirit, the assumption that the initial state is precisely known). Two questions arise:

1. is it possible to find classes of planning problems in nondeterministic domains which are still in NP?
2. when it is no longer the case that a class of planning problems is in NP, can we replace planning as satisfiability by another logical problem and therefore pursuing the planning as satisfiability paradigm consisting in solving planning problems by translating them into standard logical problems?

Section 2 contains the background about propositional logic and computational complexity. In Section 3 we consider unobservable environments; we first answer question 1, and then question 2 by showing that more generally the search for a polynomial-length plan can be expressed as a simple abduction problem. In Section 4 we introduce observations (under the assumption of full observability) and thus conditional plans, which considerably complexify the problem.

2 Background and notations

2.1 Propositional logic

Let PS be a finite set of propositional variables. $PROP_{PS}$ is the propositional language built up from PS and the connectives in the usual way. For every $X \subseteq PS$, $PROP_X$ denotes the sublanguage of $PROP_{PS}$ generated from the variables of X, only. $\top$ denotes the boolean constant that is always true, and $\bot$ the boolean constant that is always false. For every formula Σ of $PROP_{PS}$, $Var(\Sigma)$ is the set of variables occurring in Σ. Full instantiations of variables of $X \subseteq PS$ (resp. PS) (worlds) are denoted by $\vec{x}$ (resp. $\vec{\omega}$). Sometimes we identify the complete assignment $\vec{x}$ with the usual formula of $PROP_X$ formed by taking the conjunction of all X-literals as assigned by $\vec{x}$. 2^X denotes the set of all possible assignments of variables of X. If X and Y are two disjoint subsets of $PROP_{PS}$ then $(\vec{x}, \vec{y})$ is the assignment of $2^{X \cup Y}$ that assigns any variable of X like $\vec{x}$ does and any variable of Y as $\vec{y}$ does. We shall also identify finite sets of formulas with the conjunctions of their elements each time this will be convenient.

The set $PI^X_\Sigma(\Phi)$ of prime implicants modulo a formula Σ of a formula Φ over $PROP_X$ is defined (up to logical equivalence) by

$$PI^X_\Sigma(\Phi) = PI^X(\Sigma \Rightarrow \Phi) \setminus PI^X(\neg\Sigma)$$

where $PI^X(\Phi) = PI(\Phi) \cap PROP_X$ and $PI(\Phi)$ ($= PI^{PS}_\top(\Phi)$) denotes the set of prime implicants of Φ. For instance, if $\Sigma = \{a \vee b, (a \wedge c) \Rightarrow e, d \Rightarrow e\}$, then

$PI^{\{a,b,c,d,e\}}_\top(\neg\Sigma) = \{\neg a \wedge \neg b, a \wedge c \wedge \neg e, d \wedge \neg e, \neg b \wedge c \wedge \neg e\}$;

$PI^{\{a,b,c,d\}}_\Sigma(e) = \{a \wedge c, d, \neg b \wedge c\}$;

$PI^{\{a,c\}}_\Sigma(e) = \{a \wedge c\}$;

$PI^{\{a,b\}}_\Sigma(e) = \emptyset$;

$PI^{\{a,b,c,d\}}_\Sigma(a \vee b) = \{\top\}$.

2.2 Computational complexity and QBF

In this paper we refer to some complexity classes above NP and coNP, details about which can be found in [Papadimitriou, 1994]. We assume that the classes P, NP and coNP are known to the reader. $\Sigma^p_2 = \mathsf{NP}^{\mathsf{NP}}$ is the class of all languages recognizable in polynomial time by a nondeterministic Turing machine equipped with an NP oracle telling in unit time whether a given propositional formula is satisfiable or not; $\Pi^p_2 = \mathsf{co}\Sigma^p_2$. The canonical Σ^p_2-complete problem is 2-QBF (where QBF stands for "quantified boolean formula"). An instance of 2-QBF consists of a triple $\langle A, B, \Phi \rangle$ where A and B form a partition of the set of variables appearing in Φ. Such a triple is a *positive* instance of 2-QBF iff $\exists \vec{a} \in 2^A$ such that $\forall \vec{b} \in 2^B$, $(\vec{a}, \vec{b}) \models \Phi$. A characterization of the positive instances of the canonical Π^p_2-complete problem 2-$\overline{\text{QBF}}$ is obtained by reversing the quantifiers, namely: $\forall \vec{a} \in 2^A\ \exists \vec{b} \in 2^B$ such that $(\vec{a}, \vec{b}) \models \Phi$.

2-QBF (and 2-$\overline{\text{QBF}}$) instances can be characterized through prime implicants. Indeed, $\langle A, B, \Phi \rangle$ is a positive instance of 2-QBF iff $PI^A_\top(\Phi) \neq \emptyset$ [Fargier *et al.*, 2000]. Accordingly, with $\Sigma = \{a \vee b, (a \wedge c) \Rightarrow e, d \Rightarrow e\}$, $\langle \{b,c,e\}, \{a,d\}, \Sigma \rangle$ is a positive instance of 2-QBF (because $PI^{\{b,c,e\}}_\top(\Sigma) = \{b \wedge e\}$) while $\langle \{a,b,c\}, \{d,e\}, \Sigma \rangle$ is not (because $PI^{\{a,b,c\}}_\top(\Sigma) = \emptyset$). $\langle \{a\}, \{b,c,d,e\}, \Sigma \rangle$ is a positive instance of 2-$\overline{\text{QBF}}$ while $\langle \{a,b\}, \{c,d,e\}, \Sigma \rangle$ is not.

3 Unobservable domains

For the rest of the paper, the set S of possible states is the set of propositional worlds, i.e., a set of full assignments over a given set of propositional variables $V \subseteq PS$ (encoding fluents).

Each action a from the set A of actions is described by a formula $Desc(a)$ from some action language $\mathcal{L}_A$ whose semantics is defined in terms of possible transitions between states. As in Section 1, we denote by $Res(s, a)$ the set of possible results of action a in state s. An action is *nondeterministic* as soon as there is a state $s \in S$ for which $Res(s, a)$ has at least two elements. For the sake of simplicity, we assume that actions are *fully executable*[1], hence $Res(s, a)$ is never empty. We consider the special *empty action* λ defined by: for every $s \in S$, $Res(s, \lambda) = \{s\}$.

Definition 1 (planning problems)
A (finite-horizon) planning problem $\mathcal{P}$ *w.r.t. an action language $\mathcal{L}_A$ consists of a finite horizon $N \geq 1$, a propositional formula $Init$ describing the knowledge about the initial state, a finite set $A \supseteq \{\lambda\}$ of actions described in $\mathcal{L}_A$, and a propositional formula G describing the goal.*

The key definitions are the following ones:

Definition 2 (trajectories and (unconditional) plans)

- *A* trajectory *for $\mathcal{P}$ of length T is a sequence $\tau = \langle s_0, \ldots, s_T \rangle \in S^{T+1}$ of states, with $1 \leq T \leq N$. If $\tau = \langle s_0, \ldots, s_T \rangle$, then $length(\tau) = T$.*
- *A* plan *for $\mathcal{P}$ is a sequence $\pi = \langle a_0, \ldots, a_{N-1} \rangle \in A^N$ of actions. We shall sometimes write $\pi(i)$ for a_i.*

Definition 3 (compatible trajectories)
A trajectory τ for $\mathcal{P}$ is compatible *with a plan π for $\mathcal{P}$ iff $length(\tau) = N$, $s_0 \models Init$, and for each $i \in 0..N-1$, $s_{i+1} \in Res(s_i, a_i)$.*

[1] This assumption of full executability does not imply a loss of generality: it is possible to model actions with executability conditions by adding to S a new fictitious, absorbing state s_{Inexec} and by interpreting any action a on S as an action a_{new} on $S \cup \{s_{Inexec}\}$ such that:

$$Res(s, a_{new}) = \begin{cases} \{s_{Inexec}\} & \text{if } s = s_{Inexec}, \\ \{s_{Inexec}\} & \text{if } a \text{ is not executable in } s, \\ Res(s, a) & \text{otherwise.} \end{cases}$$

Definition 4 (valid (unconditional) plans)
A plan π for $\mathcal{P}$ is valid *iff for any trajectory $\tau = \langle s_0, \ldots, s_N \rangle$ for $\mathcal{P}$ that is compatible with π, we have $s_N \models G$.*

Since the empty action λ belongs to A, we can restrict ourselves to plans containing exactly N actions; indeed, there exists a valid plan for $\mathcal{P}$ if and only if there exists a valid N-steps plan for $\mathcal{P}$. While this requirement is not mandatory as a definition, it will prove helpful in the following, when valid plans will be characterized as prime implicants.

We first try to identify a sufficient condition to make the nondeterministic plan existence remain in NP. Let us define a class of planning problems as *thin* if and only if for each of its instances, the number of trajectories compatible with any plan is polynomially bounded in $|\mathcal{P}|$. We have the following complexity result:

Proposition 2 *Polynomial-size plan existence for any thin class of nondeterministic planning problems is in* NP.

This result comes easily from the fact that such planning problems with a polynomial number of trajectories, and thus a polynomial number of states, can be seen as a polynomial collection of deterministic planning problems. The thinness condition is met for instance when actions are deterministic and the number of possible initial states is polynomial.

The thinness condition is strong and most of the times it is not met. Now, what about the general case? We first state the following result which slightly generalizes the Σ_2^p -membership result by Da Costa et al. for $\mathcal{L}_A$ being nondeterministic STRIPS [da Costa *et al.*, 1997], and by Baral et al. for $\mathcal{L}_A$ being the action language $\mathcal{A}$ [Baral *et al.*, 1999].

Proposition 3 *Suppose that the action language $\mathcal{L}_A$ verifies the following property: for each action (deterministic or not) a and any pair (s, s') of states, it can be decided in polynomial time whether $s' \in Res(a, s)$. Then polynomial-size plan existence in nondeterministic environments is in Σ_2^p.*

Again, without fixing the language $\mathcal{L}_A$ it is difficult to express a completeness result. However, it is clear that Σ_2^p-completeness is obtained as soon as $\mathcal{L}_A$ is expressive enough. In particular, it is reached for $\mathcal{L}_A$ = nondeterministic STRIPS [da Costa *et al.*, 1997] and for $\mathcal{L}_A = \mathcal{A}$ [Baral *et al.*, 1999]. This means that in the general case, finding a polynomial-length plan in a nondeterministic domain is *not* polynomially reducible into a satisfiability problem (unless the polynomial hierarchy collapses).

Now, these complexity results tell us many things. Not only it tells that in the general case (i.e., as soon as the action language is expressive enough), it is hopeless translating the problem into a satisfiability problem unless the proposed translation is exponentially large, but it also gives some information about what kind of logical problem corresponds to the search of a plan in unobservable domains. The canonical Σ_2^p-complete problem being 2-QBF a planning problem under the above conditions can be rewritten as an instance of it. Now, as shown in [Fargier *et al.*, 2000], a triple $\langle A, B, \Phi \rangle$ is a *positive* instance of 2-QBF (i.e., $\exists\vec{a}\forall\vec{b}(\vec{a},\vec{b}) \models \Phi$ is true) if and only if $PI^A(\Phi) \neq \emptyset$, or in other terms, iff there is an explanation of Φ over the variables in A. This naturally leads to formulate a planning problem as an *abduction* problem.

To make things easier we fix an action language (remembering that this choice does not imply any loss of generality). We choose a simplified version of the update-based language used in [Fargier *et al.*, 2000] which can be seen as a generalization of STRIPS to nondeterministic actions, and is also very similar to the language $\mathcal{A}$ [Gelfond and Lifschitz, 1993] and to transition-based update considered in [Cordier and Siegel, 1995]. We refer to this action language as $\mathcal{HLM}$ in the following.

In $\mathcal{HLM}$, each action $a \in A$ is described by a finite set of effect rules $Desc(a) = \{\langle \varphi_i, \psi_i, Infl_i \rangle | i = 1 \ldots p(a)\}$, where φ_i and ψ_i are propositional formulas over V, and $Infl_i$ is a subset of V.

The belonging of $\langle \varphi, \psi, Infl \rangle$ to $Desc(a)$ means that if we perform a in a state s where φ holds then ψ holds at each of the possible subsequent states in $Res(s, a)$. In semantical terms, it imposes that $(\forall i = 1, \ldots, p(a))(\forall s, s' \in S)(s' \in Res(s, a)) \Rightarrow ((s \models \varphi_i) \Rightarrow (s' \models \psi_i))$. Obviously, checking that $s' \models \psi_i$ whenever $s \models \varphi_i$ for each $\langle \varphi_i, \psi_i, Infl_i \rangle \in Desc(a)$ can be done in time polynomial in $|s| + |s'| + |Desc(a)|$. The belonging of v to any $Infl_i$ means that the truth value of fluent v does not necessarily persist when the corresponding effect rule $\langle \varphi_i, \psi_i, Infl_i \rangle$ is "activated" (i.e., when $s \models \varphi_i$). For example the action of tossing a coin can be represented as

$$Desc(toss) = \{\langle \top, \top, \{Heads, Tails\} \rangle\}.$$

In $\mathcal{HLM}$, the empty action λ can be described by $Desc(\lambda) = \{\langle \top, \top, \emptyset \rangle\}$. For simplicity, we require that $Var(\psi_i) \subseteq Infl_i$ for each effect rule, i.e., if v is mentioned in the postcondition ψ_i, then it is potentially influenced by a.

Given a state s and an action a together with its description $Desc(a)$, we define

$$Eff(a, s) = \bigwedge \left\{ \psi \;\middle|\; \begin{array}{l} \langle \varphi, \psi, Infl \rangle \in Desc(a) \text{ such} \\ \text{that } s \models \varphi \end{array} \right\}$$

Intuitively, $Eff(a, s)$ is the gathering of all postconditions associated with the preconditions that are "activated" in s. We also define

$$Infl(a, s) = \bigcup \left\{ Infl \;\middle|\; \begin{array}{l} \langle \varphi, \psi, Infl \rangle \in Desc(a) \text{ such} \\ \text{that } s \models \varphi \end{array} \right\}$$

Lastly, the result of action a in state s is the set of all states s' which satisfy both conditions: (a) all gathered postconditions are satisfied, i.e., $s' \models Eff(a, s)$; (b) the truth value of all variables not appearing in any of the corresponding $Infl$ sets remains unchanged. Formally, $s' \in Res(s, a)$ iff

- $s' \models Eff(a, s)$;
- for every variable $v \in V$ not appearing in $Infl(a, s)$, s and s' give the same value to v.

It is easy to show that any action $a \in A$ is fully executable if and only if for any state $s \in S$, $Eff(a, s)$ is consistent (using that $Var(\psi_i) \subseteq Infl_i$) [Fargier *et al.*, 2000].

Let us briefly illustrate the action language $\mathcal{HLM}$ by formalizing the "bomb in the toilet" example reported in [Ferraris and Giunchiglia, 2000]. There are two packages p_1 and p_2 and one toilet t. One of the packages is armed because it contains a bomb. Dunking a package in a toilet disarms the package, but may clog the toilet. Dunking a package in a toilet is possible only if the toilet is not clogged and the package has not been previously dunked. Flushing clears the toilet. Initially, one of the packages is armed and the toilet is clear. The goal is to reach a state where all packages are disarmed. Formally, we have:

- $Init = (armed(p_1) \oplus armed(p_2)) \wedge \neg clogged(t) \wedge \neg dunked(p_1) \wedge \neg dunked(p_2)$.
- $G = \neg armed(p_1) \wedge \neg armed(p_2)$.

There are three actions here (in addition to λ): $dunk(p_1)$, $dunk(p_2)$, $flush$. They can be described in $\mathcal{HLM}$ by:

- $Desc(dunk(p_1)) = \{\langle \neg clogged(t) \wedge \neg dunked(p_1), \neg armed(p_1) \wedge dunked(p_1), \{armed(p_1), dunked(p_1), clogged(t)\}\rangle\}$,
- $Desc(dunk(p_2)) = \{\langle \neg clogged(t) \wedge \neg dunked(p_2), \neg armed(p_2) \wedge dunked(p_2), \{armed(p_2), dunked(p_2), clogged(t)\}\rangle\}$,
- $Desc(flush) = \{\langle \top, \neg clogged(t), \{clogged(t)\}\rangle\}$.

We suppose that actions behave as the empty action for the other preconditions.

The following translation (adapted from [Fargier *et al.*, 2000]) will enable us encode any $\mathcal{HLM}$ planning problem into an abduction problem.

1. *time-indexing of fluent variables*: to each variable $v \in V$ we associate $N+1$ state variables $v_0, v_1, \ldots, v_N$. Every v_t refers to the value of v at time point t – as in the planning as satisfiability framework; the set of state variables referring to time t is denoted by $StateVar_t$ and we define $StateVar = \cup_{0 \le t \le N} StateVar_t$.
2. *introduction of action variables*: we create an action variable a_t for each action $a \in A$ and each $t \in 0 \ldots N-1$, meaning that action a is performed at time t (i.e., between t and $t+1$). The set of action variables referring to time t is denoted by $ActVar_t$ and we define $ActVar = \cup_{0 \le t \le N-1} ActVar_t$.
3. *pure sequential planning*: we express that exactly one action is performed at each time point:
$Un(A,t) = \{\neg(a_t \wedge a'_t) \mid a, a' \in A, a \neq a'\} \cup \{\vee_{a \in A} a_t\}$.
$Un(N,A) = \bigcup_{t=0}^{N-1} Un(A,t)$.
4. *translation of effect rules*: for each action $a \in A$, we translate each of the effect rules $R = \langle \varphi, \psi, Infl\rangle \in Desc(a)$ following this intuitive formulation: if the precondition φ is true at time t and if action a is performed, then the postcondition ψ is true at time $t+1$. Formally,
$$Trans(R,t) = (\varphi_t \wedge a_t) \Rightarrow \psi_{t+1}$$
We let
$$Trans(a,t) = \bigwedge\{Trans(R,t) \mid R \in Desc(a)\}$$
and lastly,
$$Trans(N,A) = \bigwedge\{Trans(a,t) \mid a \in A, t = 0 \ldots N-1\}$$
5. *completion*. We must express the frame axioms, ensuring persistence of variables that are not influenced by an action. The direct translation of this step is problematic, because the variables that are not concerned with an action *depend both on the action and on the initial state*, and there are exponentially many possible initial states in the worst case. In order to cope with this problem, we proceed the other way round: we consider each state variable separately, and determine a sufficient condition under which it is not concerned by the execution of the action [Fargier *et al.*, 2000] (this is similar to Reiter's successor state axioms). For any fluent variable v, the *dependent effect rules* of v with respect to A are defined by:
$$Rules(v,A) = \bigcup_{a \in A}\{\langle a, \varphi\rangle \mid \langle \varphi, \psi, Infl\rangle \in Desc(a), v \in Infl\}$$
For any fluent variable $v \in V$, the *persistence completion* of v with respect to A at time t is defined by:
$$Comp(v,A,t) = (\bigwedge_{\langle a,\varphi\rangle \in Rules(v,A)} \neg(\varphi_t \wedge a_t)) \Rightarrow (v_t \Leftrightarrow v_{t+1})$$
We let $Comp(N,A) = \bigcup_{v \in V, t=0\ldots N-1} Comp(v,A,t)$.

Proposition 4 *Let $\mathcal{P} = \langle N, Init, A, G\rangle$ be a planning problem with horizon N for which every action $a \in A$ is fully executable and $\mathcal{HLM}$ is the action language. $\pi = \langle \pi(0), \ldots, \pi(N-1)\rangle$ is a valid plan for $\mathcal{P}$ iff*
$$Init_0 \wedge Un(N,A) \wedge Trans(N,A) \wedge Comp(N,A) \wedge (\pi(0)_0 \wedge \ldots \wedge \pi(N-1)_{N-1})$$
is consistent and
$$Init_0 \wedge Un(N,A) \wedge Trans(N,A) \wedge Comp(N,A) \wedge (\pi(0)_0 \wedge \ldots \wedge \pi(N-1)_{N-1}) \models G_N$$

(The formulas $Init_0$ and G_N are obtained by time-indexing the fluent variables in $Init$ and G in the obvious way.)

For example, $\langle dunk(p_1), flush, dunk(p_2)\rangle$ is a valid plan for the planning problem $\mathcal{P} = \langle 3, Init, A, G\rangle$. There is no valid (unconditional) plan with $N < 3$.

The above translation has a polynomial size, which enables us to conclude the following:

Proposition 5 *Plan verification under the above assumptions is* coNP*-complete.*

Lastly, we are able to express plan existence as abduction over action variables.

Proposition 6 *Let $\mathcal{P} = \langle N, Init, A, G\rangle$ be a planning problem for which every action $a \in A$ is fully executable and $\mathcal{HLM}$ is the action language. There exists a valid plan for $\mathcal{P}$ if and only if $PI^{ActVar}_{Init_0 \wedge Un(N,A) \wedge Trans(N,A) \wedge Comp(N,A)}(G_N)$ is not empty, and every prime implicant γ in $PI^{ActVar}_{Init_0 \wedge Un(N,A) \wedge Trans(N,A) \wedge Comp(N,A)}(G_N)$ identifies a valid plan π defined by: for every $t \in 0..N$, if $\gamma \models a_t$ then $\pi(t) = a_t$* [2].

[2] Note that $Un(A,t)$ ensures that there is never more than one a such that a_t is true – in case where γ does not entail any a_t for a given t then this means that the plan works whatever action is taken at time t.

Therefore, valid plans can be found using an abduction algorithm.

4 Fully observable domains

Assume now that the environment is fully observable. The action model is taken to be the same as in Section 3, however, the notion of *plan* differs, for we have now to allow for *conditional plans*.

Definition 5 ((conditional) plans)
A (conditional) plan *for a planning problem $\mathcal{P}$ is defined inductively by:*

- *for any $a \in A$, a is a plan;*
- *if π_1 and π_2 are plans, then $(\pi_1; \pi_2)$ is a plan;*
- *if π_1 and π_2 are plans and C is a propositional formula, then* `if` C `then` π_1 `else` π_2 *is a plan.*

Thus, a conditional plan is a program without loops. The *length* of a plan π is the length of its longest branch: $length(\pi)$ is defined inductively by

- if $a \in A$ then $length(a) = 1$;
- $length(\pi_1; \pi_2) = length(\pi_1) + length(\pi_2)$;
- $length($`if` C `then` π_1`else` $\pi_2)$ $= \max(length(\pi_1), length(\pi_2))$.

We also need to extend the previous notions of compatible trajectories and valid plans.

Definition 6 (compatible trajectories)
A trajectory $\tau = \langle s_0, \ldots, s_T \rangle$ *for $\mathcal{P}$ is* compatible *with a conditional plan π for $\mathcal{P}$ iff (1) $s_0 \models Init$ and (2) one of the following conditions holds:*

1. *$\pi \in A$ and $T = 1$ and $s_1 \in Res(\pi, s_0)$;*
2. *$\pi = \pi_1; \pi_2$ and τ can be decomposed in two subtrajectories $\tau_1 = \langle s_0, \ldots, s_i \rangle$, $\tau_2 = \langle s_i, \ldots, s_T \rangle$, such that τ_1 is compatible with π_1, and τ_2 is compatible with π_2;*
3. *$\pi =$* `if` C `then` π_1 `else` π_2 *and*
   - *either $s_0 \models C$ and τ is compatible with π_1,*
   - *or $s_0 \models \neg C$ and τ is compatible with π_2.*

Definition 7 (valid conditional plans)
A plan π is a valid plan *for $\mathcal{P}$ iff $length(\pi) \leq N$ and every trajectory $\tau = \langle s_0, \ldots, s_T \rangle$ which is compatible with π is such that $s_T \models G$.*

The following conditional plan is valid for the planning problem encoding the "bomb in the toilet" scenario with horizon $N = 1$.

```
if    armed(p1)
then  dunk(p1)
else  dunk(p2)
```

As for unobservable environments, we get the following result for thin classes of planning problems: polynomial-size branching plan existence for any thin class of nondeterministic planning problems under the assumption of full observability is in NP. It is easy to show that under this thinness assumption, if a planning problem has a valid plan, then it has a polynomial-sized valid plan.

In fully observable domains, the connection between planning and abduction is less direct. Nevertheless, a valid plan can be computed by the following backward algorithm which is reminiscent of dynamic programming as well as goal regression. We start with the goal G and we compute, for each action $a \in A$, the set $\Gamma(a, G)$ of all minimal conditions γ under which the execution of a certainly leads to a state satisfying G:

Definition 8 (minimal sufficient success conditions)
Let G be a goal and a a fully executable action. A consistent conjunction of fluent literals γ is a sufficient success condition *for a and G iff for any pair of states (s, s') such that $s \models \gamma$ and $s' \in Res(a, s)$, we have $s' \models G$. γ is a* minimal sufficient success condition *for a and G iff it is a sufficient success condition for a and G and no proper subconjunction of γ is. We denote by $\Gamma(a, G)$ the set of all minimal sufficient success conditions for a and G.*

When actions are described using the action language $\mathcal{HLM}$, such conditions can be computed abductively. Clearly, γ is a sufficient success condition for a and G if and only if a is a valid plan for the planning problem $\langle 1, \gamma, \{a\}, G \rangle$. Using now Proposition 4 we get the following: γ is a sufficient success condition for a and G if and only if $\gamma_0 \wedge Un(1, \{a\}) \wedge Trans(1, \{a\}) \wedge Comp(1, \{a\}) \wedge a_0$ is consistent, and $\gamma_0 \wedge Un(1, \{a\}) \wedge Trans(1, \{a\}) \wedge Comp(1, \{a\}) \wedge a_0 \models G_1$.

We are now in position to give an abductive characterization of minimal sufficient success conditions.

Proposition 7 *In the $\mathcal{HLM}$ language, we have* $\Gamma(a, G) = \{\gamma \mid \gamma_0 \in PI^{StateVar_0}_{Trans(1,\{a\}) \wedge COMP(1,\{a\}) \wedge a_0}(G_1)\}$.

Clearly, from any state satisfying $\Gamma(a, G)$, there is a valid plan of length 1 (namely, a). Now, let

$$\Gamma_A(G) = \bigvee_{a \in A} \Gamma(a, G).$$

From any state satisfying $\Gamma_A(G)$ there is a valid plan $step(A, G)$ of length 1, namely

```
if    Γ(a1, G)
then  a1
else  if    Γ(a2, G)
      then  a2
      else  if
                  (...)
                        else  if    Γ(an, G)
                              then  an
                              [else λ]
```

The process is now iterated, taking $\Gamma_A(G)$ as a new goal. This leads to the following algorithm, whose output is a conditional plan.

Start
$t := N$;
$Goal[N] := G$;
$\pi = \lambda$;
While $t > 0$ **and** $Init \not\models Goal[t]$ **do**
  $t := t - 1$;
  $\pi := (step(A, Goal[t + 1]); \pi)$
  $Goal[t] := \Gamma_A(Goal[t + 1])$
end while
If $Init \models Goal[t]$
then return π
else return `failure`
End

Proposition 8 *In the $\mathcal{HLM}$ action framework, if the above algorithm returns* `failure`, *then there is no valid plan for $\mathcal{P}$. If it returns a plan π, then π is a valid plan for $\mathcal{P}$.*

5 Related work and conclusion

In this paper, we have reported some general complexity results about plan verification and existence in deterministic and nondeterministic domains. We have also shown how planning under unobservability can be encoded as an abduction problem when actions are described using the simple language $\mathcal{HLM}$. Our Proposition 6 allows plan generation in unobservable domains with nondeterminism via an abduction algorithm. Specifically, some recent algorithms for computing incrementally *some* selected prime implicants (and not all of them) can be used here with profit [Palopoli *et al.*, 1999]. If actions have associated costs, a cost-based abduction algorithm may be used to find the cheapest plan (in particular, the shortest plan if any action has a unit weight except the empty action, whose cost is null). We plan to implement and test several abduction-based planning algorithms.

Our work can be related to many approaches to nondeterministic planning, considering both the representational aspects and the computational ones. The closest work is Fargier et al. [Fargier *et al.*, 2000]. The action model considered here is a restricted version of the one presented in this paper. Specifically, concurrency is not allowed here[3]. A major difference is that only one-stage decision making problems are considered in [Fargier *et al.*, 2000], not full (finite-horizon) planning ones (however the work in [Herzig *et al.*, 2001] can be seen as a multiple-stage extension of it). A line of work which is parallel to ours is the planning as satisfiability frameworks with actions languages, especially [Giunchiglia, 2000] and [Ferraris and Giunchiglia, 2000] based on the language $\mathcal{C}$ and [McCain and Turner, 1998] based on the causal theory developed by the authors. A main difference lies in the fact that, from the computational side, our framework is based on algorithms for abduction, and not directly on a consistency checker. Our work is also related to [Rintanen, 1999], where planning problems are translated to QBF and solved as such, and to [Finzi *et al.*, 2000] where an open-world planner (i.e., for unobservable environments), which makes uses of abduction and regression, is implemented in GOLOG.

[3]Note that it would not be very hard to take it into account in $\mathcal{HLM}$.

Acknowledgements

The third author has been partly supported by the IUT de Lens, and the Région Nord/Pas-de-Calais.

References

[Baral *et al.*, 1999] C. R. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in presence of incompleteness. In *Proc. of IJCAI'99*, pages 948–955, 1999.

[Bylander, 1994] T. Bylander. Complexity results for planning. *Artificial Intelligence*, 1994.

[Cordier and Siegel, 1995] M.O. Cordier and P. Siegel. Prioritized transitions for updates. In *Proc. of ECSQARU'95*, pages 142–150. LNAI 946, Springer-Verlag, 1995.

[da Costa *et al.*, 1997] C. da Costa, F. Garcia, J. Lang, and R. Martin-Clouaire. Possibilistic planning: Representation and complexity. In *Proc. of ECP'97*, pages 143–155, 1997.

[Fargier *et al.*, 2000] H. Fargier, J. Lang, and P. Marquis. Propositional logic and one-stage decision making. In *Proc. of KR'2000*, pages 445–456, 2000.

[Ferraris and Giunchiglia, 2000] P. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. of AAAI'00*, pages 748–753, 2000.

[Finzi *et al.*, 2000] A. Finzi, F. Pirri, and R. Reiter. Open-world planning in the situation calculus. In *Proceedings of AAAI'2000*, pages 754–760, 2000.

[Gelfond and Lifschitz, 1993] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, pages 301–322, 1993.

[Giunchiglia, 2000] E. Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints, and nondeterminism. In *Proc. of KR'2000*, pages 657–666, 2000.

[Herzig *et al.*, 2001] A. Herzig, J. Lang, P. Marquis, and T. Polacsek. Actions, updates, and planning. In *Proc. of IJCAI'2001*, 2001.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of ECAI'92*, pages 359–363, 1992.

[McCain and Turner, 1998] N. McCain and H. Turner. Satisfiability planning with causal theories. In *Proc. of KR'98*, pages 212–223, 1998.

[Palopoli *et al.*, 1999] L. Palopoli, F. Pirri, and C. Pizzuti. Algorithms for the selective enumeration of prime implicants. *Artificial Intelligence*, 111:41–72, 1999.

[Papadimitriou, 1994] C. Papadimitriou. *Computational complexity*. Addison–Wesley, 1994.

[Rintanen, 1999] J. Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–353, 1999.