Computational Complexity of Planning and Approximate Planning in Presence of Incompleteness

Chitta Baral, Vladik Kreinovich, and Raul Trejo
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
emails {chitta,vladik,rtrejo}@cs.utep.edu

Abstract

In the last several years the computational complexity of classical planning and HTN planning have been studied. But in both cases it is assumed that the planner has complete knowledge about the initial state. Recently, there has been proposal to use 'sensing' actions to plan in presence of incompleteness. In this paper we study the complexity of planning in such cases. In our study we use the action description language $\mathcal{A}$ proposed in 1993 by M. Gelfond and V. Lifschitz and its extensions.

The language $\mathcal{A}$ allows planning in the situations with complete information. It is known that, if we consider only plans of feasible (polynomial) length, the planning problem for such situations is NP-complete: even checking whether a given objective is attainable from a given initial state is NP-complete. In this paper, we show that the planning problem in presence of incompleteness is indeed harder: it belongs to the next level of complexity hierarchy (in precise terms, it is $\Sigma_2\mathbf{P}$-complete). To overcome the complexity of this problem, C. Baral and T. Son have proposed several approximations. We show that under certain conditions, one of these approximations - O-approximation - makes the problem NP-complete (thus indeed reducing its complexity).

1 Introduction

The action description language $\mathcal{A}$ proposed in 1993 by M. Gelfond and V. Lifschitz [Gelfond and Lifschitz, 1993] mid its successors have made it easier to understand the fundamentals (such as inertia, ramification, qualification, concurrency, sensing, etc.) involved in formalizing actions and their effects on a world, without getting into the details of particular logics. In this paper, we will be analyzing the complexity of planning based on this language and its extensions; let us, therefore, start with a brief description of this language.

1.1 Language *A:* brief reminder

In the language $\mathcal{A}$, we start with a finite list of properties (fluents) $f_1, \ldots, f_n$ which describe possible properties of a state. A *state* is then defined as a finite set of fluents, e.g., {} or $\{f_1, f_3\}$. We are assuming that we have a complete knowledge about the initial state: e.g., $\{f_1, f_3\}$ means that in the initial state, properties f_1 and f_3 are true, while all the other properties $f_2, f_4, \ldots$ are false. The properties of the initial state are described by formulas of the type

$$\text{initially } F,$$

where F is a *fluent literal,* i.e., either a fluent f_i or its negation $\neg f_i$.

To describe possible changes of states, we need a finite set of *actions.* In the language $\mathcal{A}$, the effect of each action *a* can be described by formulas of the type

$$a \text{ causes } F \text{ if } F_1, \ldots, F_m,$$

where $F, F_1, \ldots, F_m$ are fluent literals. A reasonably straightforward semantics describes how the state changes after an action:

- if before the action a, the literals $F_1, \ldots, F_m$ were true, and the domain description contains a rule according to which a causes F if $F_1, \ldots, F_m$, then this rule is *activated,* and after the action a, F becomes true; thus, for some fluent f_i, we will conclude f_i and for some other, that $\neg f_i$ holds in the next state;
- if for some fluent f_i, no activated rule enables us to conclude that f_i is true or false, this means that the action *a* does not change the truth of this fluent; therefore, f_i is true in a new state if and only if it is true in the old state.

Formally, a *domain description D* is a finite set of *value propositions* of the type initially F (which describe the initial state), and a finite set of *effect propositions* of the type "a causes *F* if $F_1, \ldots, F_m$" (which describe results of actions). A *state s* is a finite set of fluents. The *initial state SO* consists of all the fluent f_i for which the corresponding value proposition initially f_i is contained in the domain description. We say that a fluent f_i *holds* in *s* if $f_i \in s$; otherwise, we say that $\neg f_i$ holds in s. The *transition function* $Res_D(a, s)$ which describes the effect of an action a on a state *s* is defined as follows:

- we say that an effect proposition "a causes F if $F_1, \ldots, F_m$" is *activated* in a state s if all m fluent literals $F_1, \ldots, F_m$ hold in s;
- we define $V_D^+(a, s)$ as the set of all fluents fi for which a rule "a causes f_i if $F_1, \ldots, F_m$" is activated in s;
- similarly, we define $V_D^-(a, s)$ as the set of all fluents f_i for which a rule "a causes $\neg f_i$ if $F_1, \ldots, F_m$" is activated in s;
- if $V_D^+(a, s) \cap V_D^-(a, s) \neq \emptyset$, we say that the result of the action a is *undefined;*
- if the result of the action a is not undefined in a state s (i.e., if $V_D^+(a, s) \cap V_D^-(a, s) = \emptyset$), we define $Res_D(a, s) = (s \cup V_D^+(a, s)) \setminus V_D^-(a, s)$.

A *plan p* is defined as a sequence of actions $[a_1, \ldots, a_m]$. The *result* $Resr_D(p, s)$ of applying a plan p to the initial state so is defined as

$$Res_D(a_m, Res_D(a_{m-1}, \ldots, Res_D(a_1, s_0)) \ldots)).$$

The *planning problem* is: given a domain D and a desired fluent literal F, to find a plan which leads to the state in which F is true. (More complicated goals can be reformulated in these terms.)

1.2 An extension of language $\mathcal{A}$ which describes sensing actions: brief reminder

The language A describes planning in the situations with *complete* information, when we know exactly which fluents hold in the initial state and which don't. In real life, we often have only *partial* information about the initial state: about some fluents, we know that they are true in the initial state, about some other fluents, we know that they are false in the initial state; and it is also possible that about some fluents, we do not know whether they are initially true or false. In such situations, the required action depends on the state: e.g., if we want the door closed, the required action depends on whether the door was initially open (then we close it), or it was already closed (then we do nothing). Therefore, for these situations, we must include *sensing actions* - e.g., an action *checki* which checks whether the fluent /, holds in a given state - to our list of actions, and allow *conditional* plans, i.e., plans in which the next action depends on the result of the previous sensing action.

Some fluents may be difficult to detect, so we may have sensing actions only for *some* fluents; some real-life sensing actions may sense *several* fluents at a time. In view of these possibilities, the precise formulation of this language is as follows[1]. In the domain description D, in addition to value propositions and effect propositions, we can also have *sensing* propositions, of the type "a determines f_i". A *k-state* is defined as pair $\langle s, \Sigma \rangle$, where s is the *actual* state, and Σ is the set of all possible states which are consistent with our current knowledge. Initially, the set Σ_0 consists of all the states s for which:

- a fluer f_i is true $(f_i \in s)$ if the domain description D contains the proposition "initially f_i";
- a fluent fi is false $(f_i \notin s)$ if the domain description D contains the proposition "initially $\neg f_i$".

If neither the proposition "initially f_i", nor the proposition "initially $\neg f_i$" are in the domain description, then Σ_0 contains states with f_i true *and* with f_i false. The actual initial state s_0 can be any state from the set Σ_0. The transition function is defined as follows:

- for proper *(non-sensing)* actions, $\langle s, \Sigma \rangle$ is mapped into $\langle Res_D(a, s), Res_D(a, \Sigma) \rangle$, where:
  - *Reso(a, s)* is defined as in the case of complete information, and $Res_D(a, \Sigma) = \{Res_D(a, s') \mid s' \in \Sigma\}$.
- for a *sensing* action a which senses fluents $f_1, \ldots, f_k$ – i.e., for which sensing propositions "a determines f_i" belong to the domain D ~ the actual state s remains unchanged while Σ is down to only those states which have the same values of f_i as s: $\langle s, \Sigma \rangle \to \langle s, \Sigma' \rangle$, where

$$\Sigma' = \{s' \in \Sigma \mid \forall i (1 \le i \le k \to (f_i \in s' \leftrightarrow f_i \in s))\}$$

In the presence of sensing, an action plan is no longer a pre-determined sequence of actions: if one of these actions is sensing, then the next action may depend on the result of that sensing. In general, the choice of a next action may depend on the results of all previous sensing actions. Such an action plan is called *conditional.*

Examples have shown that adding sensing actions increases the computational complexity of the problem. In this paper, we show that the corresponding planning problem is indeed harder: it belongs to the next level of complexity hierarchy (in precise terms, it is Σ_2P-complete).

1.3 The notion of a O-approximation

To overcome the complexity of this problem, C. Baral and T. Son have proposed several approximations, whose plans are always correct but which can miss a plan. The first approximation - called *O-approximation* - is as follows: An *a-state* (approximate state) s is a finite set of fluent literals (i.e., fluents and their negations). The *initial a-state* so consists of all the fluent literals F for which the corresponding value proposition "initially F" is contained in the domain description. We say that:

- a fluent *fi* if *true* in s is $f_i \in s$;
- a fluent *fi* if *false* in s is $\neg f_i \in s$;
- a fluent *fi* if *unknown* in s is neither $f_i \in s$, not $\neg f_i \in s$.

The *transition function Resi)(a,s)* which describes the effect of a proper action a on an a-state s is defined as follows:

[1]The formulation given here is based on earlier work of formalizing sensing actions in [Moore, 1985; Scherl and Levesqne, 1993].

- we say that an effect proposition "a causes F if $F_1, \ldots, F_m$" is *activated* in an a-state *s* if all m fluent literals $F_1, \ldots, F_m$ hold in s;
- we say that an effect proposition "a causes *F* if $F_1, \ldots, F_m$" is *possibly activated in* an a-state *s* if all m fluent literals $F_1, \ldots, F_m$ possibly hold in *s* (i.e., are either true, or unknown in s);
- we define $V_D(a, s)$ as the set of all fluent literals *F* for which a rule "a causes *F* if $F_1, \ldots, F_m$" is activated in S, and no rule "$\neg a$ causes *F* if $F_1, \ldots, F_m$" is possibly activated in *s;*
- we then define $Resr_D(a,s)$ as

$$\{F \mid (F \in s \,\&\, \neg F \notin V_D(a,s)) \vee F \in V_D(a,s)\}.$$

For *sensing* actions, the result of applying a to an a-state *s* simply means adding, to the a-state, the fluent literals which turned out to be true as a result of this sensing action.

2 Results

2.1 What kind of planning problems we are interested in

Informally speaking, we are interested in the following problem:

- *given* a domain description (i.e., the description of the initial state and of possible consequences of different actions) and a goal (i.e., a fluent which we want to be true),
- *determine* whether it is possible to achieve this goal (i.e., whether there exists a plan which achieves this goal).

We are interested in analyzing the *computational complexity* of the planning problem, i.e., analyzing the computation time which is necessary to solve this problem.

Ideally, we want to find cases in which the planning problem can be solved by *a, feasible* algorithm, i.e., by an algorithm $\mathcal{U}$ whose computational time $t_{\mathcal{U}}(w)$ on each input w is bounded by a polynomial $p(|w|)$ of the length $|w|$ of the input w: $t_{\mathcal{U}}(x) \leq p(|w|)$ (this length can be measured bit-wise or symbol-wise. Problems which can be solved by such *polynomial-time* algorithms are called problems from the class $\mathbf{P}$ (where $\mathbf{P}$ stands for polynomial-time). If we cannot find a polynomial-time algorithm, then at least we would like to have an algorithm which is as close to the class of feasible algorithms as possible.

In short, we are interested in restricting the time which it takes to check whether the planning problem is solvable. This interest is justified because in planning applications we often want the resulting plan to be produced in real time, and if it is not possible to produce such a plan, we would like to know about this impossibility as early as possible, so that we will be able to add new actions (or simply give up). Since we are operating in a time-bounded environment, we should worry not only about the time for *computing* the plan, but we should also worry about the time that it takes to actually *implement* the plan. If an action plan consists of a sequence of 2^{2^n} actions, then this plan is not feasible. It is therefore reasonable to restrict ourselves to *feasible* plans, i.e., by plans u whose length $|u|$ (= number of actions in it) is bounded by a polynomial $p(|w|)$ of the input w. With this feasibility in mind, we can now formulate the above planning problem in precise terms:

- *given:* a polynomial $p(n) \geq n$, a domain description *D* (i.e., the description of the initial state and of possible consequences of different actions) and a goal / (i.e., a fluent which we want to be true),
- *determine* whether it is possible to feasibly achieve this goal, i.e., whether there exists a feasible plan *u* (with $|u| \leq p(|D|)$) which achieves this goal.

We are interested in analyzing the *computational complexity* of this planning problem.

2.2 Complexity of the planning problem for situations with complete information

For situations with complete information, the above planning problem is **NP-complete**:

Theorem 1. *For situations with complete information, the planning problem is* **NP** *-complete.*

Comments.

- This result is similar to the result of Liberatore [Liberatore, 1997]. The main difference is that Liberatore considers *arbitrary* queries from the language *A,* while we only consider queries about the existence of a feasible action plan.
- The result of Liberatore is preceded by the results of Erol et al [Erol et al., 1995] where they study complexity of STRIPS. Here we use $\mathcal{A}$ and its extensions instead of STRIPS as to the best of our knowledge there has not been any *formal treatment* of extensions of STRIPS dealing with sensing actions.
- For lack of space we are not able to present all the proofs in this paper.
- The problem remains NP-complete even if we consider the planning problems with a fixed finite number of actions: even with *two* actions. If we only allow a single action, then there is no planning any more: the only possible plan is, in any state, to apply this only possible action and check whether we have achieved our goal yet; the corresponding "planning" problem is, of course, solvable in polynomial time.

2.3 Useful complexity notions

For situations with incomplete information, the planning problem is more complicated - actually, belongs to the next levels of polynomial hierarchy; see the exact results below. For precise definitions of the polynomial hierarchy, see, e.g., [Papadimitriou, 1994]. Crudely speaking, a decision problem is a problem of deciding whether a given input w satisfies a certain property P (i.e., in set-theoretic terms, whether it belongs to the corresponding set $S = \{w \mid P(w)\}$).

- A decision problem belongs to the class $\mathbf{P}$ if there is a feasible (polynomial-time) algorithm for solving this problem.
- A problem belongs to the class $\mathbf{NP}$ if the checked formula $w \in S$ (equivalently $P(w)$) an be represented as $\exists u P(u, w)$, where $P(u, w)$ is a feasible property, and the quantifier runs over words of feasible length (i.e., of length limited by some given polynomial of the length of the input). The class $\mathbf{NP}$ is also denoted by $\Sigma_1\mathbf{P}$ to indicate that formulas from this class can be defined by adding 1 existential quantifier (hence Σ and 1) to a polynomial predicate $(\mathbf{P})$.
- A problem belongs to the class $\mathbf{coNP}$ if the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\forall u P(u, w)$, where $P(u, w)$ is a feasible property, and the quantifier runs over words of feasible length (i.e., of length limited by some given polynomial of the length of the input). The class $\mathbf{coNP}$ is also denoted by $\Pi_1\mathbf{P}$ to indicate that formulas from this class can be defined by adding 1 universal quantifier (hence Π and 1) to a polynomial predicate (hence $\mathbf{P}$).
- For every positive integer k, a problem belongs to the class $\Sigma_k\mathbf{P}$ if the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\exists u_1 \forall u_2 \ldots P(u_1, u_2, \ldots, u_k, w)$, where $P(u_1, \ldots, u_k, w)$ is a feasible property, and all k quantifiers run over words of feasible length (i.e., of length limited by some given polynomial of the length of the input).
- Similarly, for every positiveinteger k, a problem belongs to the class $\Pi_k\mathbf{P}$ if the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\forall u_1 \exists u_2 \ldots P(u_1, u_2, \ldots, u_k, w)$, where $P(u_1, \ldots, u_k, w)$ is a feasible property, and all k quantifiers run over words of feasible length (i.e., of length limited by some given polynomial of the length of the input).
- All these classes $\Sigma_k\mathbf{P}$ and $\Pi_k\mathbf{P}$ are subclasses of a larger class $\mathbf{PSPACE}$ formed by problems which can be solved by a polynomial-*space* algorithm. It is known (see, e.g., [Papadimitriou, 1994]) that this class can be equivalently reformulated as a class of problems for which the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\forall u_1 \exists u_2 \ldots P(u_1, u_2, \ldots, u_k, w)$, where the number of quantifiers k is bounded by a polynomial of the length of the input, $P(u_1, \ldots, u_k, w)$ is a feasible property, and all k quantifiers run over words of feasible length (i.e., of length limited by some given polynomial of the length of the input).

A problem is called *complete* in a certain class if. crudely speaking, this, is the toughest problem in this class (so that any other general problem from this class can be reduced to it by a feasible-time reduction). It is still not known (1998) whether we can solve any problem from the class $\mathbf{NP}$ in polynomial time (i.e., in precise terms, whether $\mathbf{NP}=\mathbf{P}$). However, it is widely believed that we cannot, i.e., that $\mathbf{NP}\neq\mathbf{P}$. It is also believed that to solve a $\mathbf{NP}$-complete or a $\mathbf{coNP}$-plete problem, we need exponential time $\approx 2^n$, and that solving a complete problem from one of the second-level classes $\Sigma_2\mathbf{P}$ or $\Pi_2\mathbf{P}$ requires more computation time than solving NP-complete problems (and solving complete problems from the class $\mathbf{PSPACE}$ takes even longer).

2.4 Complexity of the planning problem for situations with incomplete information: situations with no sensing actions

Let us start our analysis with the case of no sensing.

Theorem 2. *For situations with incomplete information and without sensing, the planning problem is $\Sigma_2\mathbf{P}$-complete.*

Proof. The problem is to check the existence of a feasible-length action plan u_1 for which, for every set of values u_2 of the unknown fluents u_1 is successful, i.e., we check whether $\exists u_1 \forall u_2 P(u_1, u_2, w)$. Once we know u_1 and u_2 (i.e., once we know the initial state and the actions), we can determine, step-by-step, all following states, and thus check, in polynomial time, whether in the final state, the desired predicate is true. So, $\mathcal{P} \in \Sigma_2\mathbf{P}$.

To show that $\mathcal{P}$ is complete, we reduce, to $\mathcal{P}$, a known complete propositional problem of checking $\exists x_1 \ldots \exists x_m \forall x_{m+1} \ldots \forall x_n F$, ($x_i$ are propositional variables, F is a propositional formula). To reduce it to $\mathcal{P}$, we first *parse* F, i.e., we represent computing *F* as a sequence of elementary steps, on each of which we apply &, V, or ¬ to compute the intermediate results $x_{n+1}, \ldots, x_N$: e.g., to compute $(x_1 \vee x_2) \& (x_1 \vee \neg x_2)$, we compute $x_3 := x_1 \vee x_2$, etc. In our planning problem, we take two actions *a* and a^-, and fluents $x_1, \ldots, x_N$, $s_0, s_1, \ldots, s_N$ (meaning: s_i is true iff time = i). Initially, s_0 is true, all other s, are false; x_N is false, all other x_i are unknown; goal: $x_N =$"true". In the firs$m$ moments of time, we select variables $x_1, \ldots, x_m$: *a* selects x_i, a^- selects $\neg x_i$: "a causes x_i i s_{i-1}" a m e f o a^-);s o , every action increases time by one: e.g., *a* causes s_i if s_{i-1} and causes $\neg s_{i-1}$ if s_{i-1}. In moments $k = n+1, \ldots, N$, we "compute" x_k: e.g., if $x_k := x_f \& x_s$, then a causes x_k if s_{k-1}, x_f, x_s,, causes $\neg x_k$ if $s_{k-1}, \neg x_f$, and causes $\neg x_k$ if $s_{k-1}, \neg x_s$ (+ rules which increase time by 1). A plan exists iff there exist values $x_1, \ldots, x_m$ for which, for all $x_{m+1}, \ldots, x_n$, F is true. The reduction proves that $\mathcal{P}$ is complete.

The problem remains $\Sigma_2\mathbf{P}$-complete even if we consider the planning problems with a fixed finite number of actions: even with two actions.

Theorem 3. *For situations with incomplete information and without sensing, the 0-approximation to the planning problem is NP-complete.*

In other words, the use of O-approximation cuts off one level from the complexity. So, for this problem, 0-approximation is indeed computationally very efficient.

This reduction is in good accordance with our intuitive understanding of this problem and its O-approximation:

- In the case oi complete information, to represent a state, we must know which fluents are true and which are false. Therefore, a state can be uniquely described by a subset of the set of all the fluents - namely, the subset consisting of those fluents which are true in this state. The total number of states is therefore equal to the total number of such subsets, i.e., to 2^F (where F is the total number of fluents).
- In the case of incomplete information, we, in general, do not know which states the system is. So, a state of our knowledge (called a *k-state* in [Son and Baral, 1998]) can be represented by a *set* of possible complete-information states. Therefore, the number of all possible k-states is equal to the number of all possible subsets of the set of all complete-information states, i.e., to 2^{2^F}.
- In O-approximation, an a-state is represented by stating which fluents are true, which are false, and which are unknown. For each of F fluents, there are three different possibilities, so totally, in this approximation, we have 3^F possible a-states.

So, going from a full problem to its O-approximation decreases the number of possible "states" from doubly exponential 2^{2^F} to singly exponential 3^F. Since planning involves analyzing different possible states, it is no wonder that for O-approximation, the computation time should also be smaller. Again, this argument is *not* a proof of Theorem 3, but this argument makes the result of Theorem 3 intuitively reasonable.

2.5 Complexity of the planning problem for situations with incomplete information: situations with sensing

Let us now consider what will happen if we allow sensing actions. If we allow unlimited sensing, then the situation changes radically-, the planning problem becomes so much more complicated that O-approximation is not helping anymore:

Theorem 4. *For situations with incomplete information and with sensing, the planning problem is PSPACE-complete.*

Theorem 5. *For situations with incomplete information and with sensing, the O-approximation to the planning problem is PSPACE-complete.*

The proofs are similar to [Littman, 1997]. Both the planning problem itself and its O-approximation remain PSPACE-complete even if we consider the planning problems with a fixed finite number of actions: even with two proper actions and a single sensing action which reveals the truth value of only one fluent - but we are allowed to repeat this sensing action at different moments of time.

In many real life control and planning situations, it is desirable to monitor the environment continuously, and to make sensing actions all the time. However, this necessity is caused by the fact that in many real-life situations, the consequences of each action are only statistically known, so we need to constantly monitor the situation to find out the actual state. In this paper, we consider the situations in which the result of each action is uniquely *determined* by this action and by the initial state. In such idealized situations, there is no such need for a constant monitoring. It therefore makes sense to allow only a limited repetition of sensing actions in an action plan. With such a limitation, the complexity of planning drops back, and O-approximation starts helping again:

Definition 1. *Let k be a positive integer.*

- *We say that a sensing action is k-limited if it reveals the values of no more than k fluents.*
- *We say that an action plan is k-bounded if it has no more than k sensing actions.*

Theorem 6. *For any given k, for situations with incomplete information and with k-limited sensing actions, the problem of checking the existence of a k-bounded action plan is* $\Sigma_2 P$*-complete.*

Theorem 7. *For any given k, for situations with incomplete information and with k-limited sensing actions, the problem of checking the existence of a k-bounded 0-approximation action plan is NP-complete.*

Comments.

- The same result holds if instead of assuming that k is a constant, we allow k to grow as $\sqrt{\log(|D|)}$ (i.e., as a square root of the logarithm of the length of the input).
- A difficulty with the general situation with incomplete information comes from the fact that we do not know the *exact* states, i.e., we do not know the values of *all* the fluents. It is therefore reasonable to analyze the situations with *full sensing, i.e.,* situations In which, for every fluent fi, we have a sensing action $check_i$ which reveals the value of this fluent. Full sensing does make the planning problem simpler, although not that simpler so that 0-approximation will help:

Theorem 8. *For situations with incomplete information and with full sensing, the planning problem is* $\Pi_2 P$*-complete.*

Theorem 9. *For situations with incomplete information and with full sensing, the O-approximation to the planning problem is* $\Pi_2 P$*-complete.*

These results can be represented by the following table:

	exact planning	0-approximation
complete information	**NP**-complete	**NP**-complete
partial informat-, ion, no sensing	Σ_2**P**-complete	**NP**-complete
limited number of sensing actions	Σ_2**P**-complete	**NP**-complete
unlimited number of sensing actions	**PSPACE**-complete	**PSPACE**-complete
partial informat-ion, full sensing	Π_2**P**-complete	Π_2**P**-complete

unlimited number
of sensing actions
PSPACE-complete
↙ ↘
full sensing — Π_2**P**-complete
limited sensing — Σ_2**P**-complete
↙
O-approximation or
complete information
NP-complete

2.6 Auxiliary result: 1-approximation is coNP-complete

In addition to O-approximation, the authors of [Baral and Son, 1997; Son and Baral, 1998] considered other types of approximations, including the so-called *1-approximation.* In 1-approximation, partial states are defined in the same manner as for O-approximation: i.e., as lists of fluents and their negations. However, the result of a (proper) action *a* on an a-state *s* is defined differently: in this new approximation, a fluent literal *F* (fluent or its negation) is true after applying *a* to *s* if and only if *F* is true in all possible complete states complementing *s.* Then, as a new a-state $Resr_D(a,s)$, we take the set of all fluent literals which are true after applying *a.*

In this section, we will show that this new definition increases the computational complexity of an approximation. Namely, while for O-approximation, computing the next a-state *ResD(a,s)* was a polynomial-time procedure, for 1-approximation, computing the next state is already a coNP-complete problem:

Theorem 10. (1-approximation) *The problem of checking, for a given a-state s, for a given action a, and for a given fluent f, whether f is true in* $Res_D\{a, s)_\}$ *is* coNP-*complete.*

Comments.

- An **ω-approximation** is defined in a similar manner, except that in an **ω-approximation,** the result $Res_D(a, s)$ is defined not after a single action a, but after a sequence of proper actions between two sensing actions. In the particular case when there is exactly one proper action between the two sensing actions, ω-approximation reduces to 1-approximation. Therefore, ω-approximation is also at least as complicated as coNP-complete *problems.*
- These *results show that if we want* an *approximation to decrease the computational complexity* of the planning problem, *then (at least from* the viewpoint of the worst-case complexity) 0-approximation is preferable to 1-approximation and w-approximation.

References

[Baral and Son, 1997] C. Baral and T. Son, "Approximate reasoning about actions in presence of sensing and incomplete information", In: *Proc. of International Logic Programming Symposium (ILPS^l),* 1997, pp. 387-401.

[Erol et al., 1995] K. Erol, D. S. Nau, and V. S. Subrahmanian, "Complexity, decidability and undecidability results for domain-independent planning", *Artificial Intelligence,* 1995, Vol. 76, pp. 75-88 (detailed proofs are given in the University of Maryland Technical Report CS-TR-2797 (also listed as UMIACS-TR-91-154 and SRC-91-96).

[Gelfond and Lifschitz, 1993] M. Gelfond and V. Lifechitz, "Representing actions and change by logic programs", */. of Logic Programming,* 1993, Vol. 17, pp. 301-322.

[Liberatore, 1997] P. Liberatore, "The complexity of the language *A", Electronic Transactions on Artificial Intelligence,* 1997, Vol. 1, pp. 13-28 (http://www.ep.iiu.se/ej/etai/1997/02).

[Littman, 1997] M. Littman, "Probabilistic propositional planning: representation and complexity", *AAAI'97,* pp. 748-754.

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

[Papadimitriou, 1994] C. H. Papadimitriou, *Computational Complexity,* Addison-Wesley, Reading, MA, 1994.

[Scherl and Levesque, 1993] R. Scherl and H. Levesque, "The frame problem and knowledge producing actions." In *KR 93,* pages 689-695, 1993.

[Son and Baral, 1998] T. Son, and C. Baral, *Formalizing sensing actions - a transition function based approach,* University of Texas at El Paso, Department of Computer Science, Technical Report, 1998 (http://cs.utep.edu/chitta/chitta.html).