Computational Complexity of Planning Without Sensing, Planning With Sensing, and Approximate Planning

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

One of the main problems in knowledge representation in to describe actions and their effects. One of the most successful formalisms for describing actions is the language $\mathcal{A}$ proposed in 1993 by M. Gelfond and V. Lifschitz. This language describes planning in the situations with complete information. It is known that 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 real life, we often have only partial information about the situation. In this case, it is reasonable to add measurements ("sensing actions") to the list of possible actions. 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_2**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 – 0-approximation – makes the problem **NP**-complete (thus indeed reducing its complexity).

1 Introduction

One of the main problems in knowledge representation in to describe actions and their effects. One of the most successful formalisms for describing actions is the language $\mathcal{A}$ proposed in 1993 by M. Gelfond and V. Lifschitz [2]. In this paper, we will be analyzing the complexity of planning based on this language and on its extensions; let us, therefore, start with a brief description of this language.

1.1 Language $\mathcal{A}$: 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

$$\textsf{initially}\ F,$$

where F is a *fluent expression* (*atom*), 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

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

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

- if before the action a, atoms $F_1, \ldots, F_m$ 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 fluents 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 $\textsf{initially}\ F$ (which describe the initial state), and a finite set of *effect propositions* of the type "a $\textsf{causes}$ F $\textsf{if}$ $F_1, \ldots, F_m$" (which describe results of actions). A *state* s is a finite set of fluent names. The *initial state* s_0 consists of all the fluents names f_i for which the corresponding value proposition $\textsf{initially}\ f_i$ is contained in the domain description. We say that a fluent f_i *holds* in s is $f_i \in s$; 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 $\textsf{causes}$ F $\textsf{if}F_1, \ldots, F_m$" is *activated* in a state s if all m fluent expressions $F_1, \ldots, F_m$ hold in s;

- we define $V_D^+(a,s)$ as the set of all fluent names f_i 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 fluent names 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* of applying a plan p to the initial state s_0 is defined as

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

The *planning problem* is: given a domain D and a desired fluent expression F, to find a plan which leads to the state in which F is true.

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

The language $\mathcal{A}$ 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 $check_i$ which checks whether the fluent f_i 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. In the domain description D, in addition to value propositions and effect propositions, we can also have *sensing* propositions, of the type "a $\mathsf{determines}$ f_i". A *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 fluent f_i is true ($f_i\in S$) if the domain description D contains the proposition "$\mathsf{initially}$ f_i";

- a fluent f_i is false ($f_i \not\in s$) 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, $\langle s, \Sigma\rangle$ changes into $\langle Res_D(a, s), Res_D(a, \Sigma)\rangle$, where:
  - $Res_D(a, s)$ 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:

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

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 $\Sigma_2\mathbf{P}$-complete).

1.3 The notion of a 0-approximation

To overcome the complexity of this problem, C. Baral and T. Son have proposed several approximations. The first approximation – called 0-*approximation* – is as follows: A *k-state* s is a finite set of fluent expressions (i.e., fluents and their negations). The *initial state* s_0 consists of all the fluent expressions F for which the corresponding value proposition "initially F" is contained in the domain description. We say that:

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

The *transition function* $Res_D(a, s)$ which describes the effect of a proper action a on a k-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 k-state s if all m fluent expressions $F_1, \ldots, F_m$ hold in s;
- we define $V_D(a, s)$ as the set of all fluent expressions F for which a rule "a causes F if $F_1, \ldots, F_m$" is activated in s;
- if $V_D(a, s)$ contains both f_i and $\neg f_i$ for some fluent f_i, then 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, we define $Res_D(a, s)$ as follows:

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

For *sensing* actions, the result of applying a to a k-state s simply means adding, to the k-state, the fluent expressions 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: $t_{\mathcal{U}}(x) \le p(|w|)$ (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 **P** (where **P** stands for *p*olynomial-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 in 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 f (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: a brief reminder

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 [3]. The main difference is that Liberatore considers *arbitrary* queries from the language $\mathcal{A}$, while we only consider queries about the existence of a feasible action plan.
- For readers' convenience, all the proofs are placed in the special Proofs section.
- As we will see from the proof, 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 Towards complexity of the planning problem for situations with incomplete information: a brief description of the necessary 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., [4]. 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 **P** if there is a feasible (polynomial-time) algorithm for solving this problem.

- A problem belongs to the class **NP** if the checked formula $w \in S$ (equivalently, $P(w)$) can 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 **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 (**P**).

- A problem belongs to the class **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 **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 **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 positive integer 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 **PSPACE** formed by problems which can be solved by a polynomial-*space* algorithm. It is known (see, e.g., [4]) 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 **NP** in polynomial time (i.e., in precise terms, whether **NP**=**P**). However, it is widely believed that we cannot, i.e., that **NP**$\neq$**P**. It is also believed that:

- to solve a **NP**-complete or a **coNP**-complete problem, we need exponential time $\approx 2^n$;
- to solve a complete problem from one of the second-level classes $\Sigma_2\mathbf{P}$ or $\Pi_2\mathbf{P}$, we need doubly exponential time $\approx 2^{2^n}$;
- to solve a complete problem from the class **PSPACE**, we need time which grows faster than any iteration of exponential functions.

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.*

As we will see from the proof, 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 0-approximation cuts off one level from the complexity, i.e., crudely speaking, replaces doubly exponential worst-case time complexity with exponential time complexity. So, for this problem, 0-approximation is indeed computationally very efficient.

This reduction from a doubly exponential to simply exponential is in good accordance with our intuitive understanding of this problem and its 0-approximation:

- In the case of 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 [5, 6]) can be represented by a *set* of possible complete-information states. Therefore, the set 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 0-approximation, a k-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 states.

So, going from a full problem to its 0-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 0-approximation, the computation time should also be exponentially smaller. Again, this argument is *not* a proof of Theorem 3 (the proof is given in the last section), 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 0-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* 0-*approximation to the planning problem is* **PSPACE**-*complete.*

As we will see from the proof, both the planning problem itself and its 0-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 much 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 0-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. *Let k be a positive integer. For situations with incomplete information and with k-limited sensing actions, the problem of checking the existence of a k-bounded action plan is $\Sigma_2\mathbf{P}$-complete.*

Theorem 7. *Let k be a positive integer. 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 $\mathbf{NP}$-complete.*

Comments.

- As we can see from the proof, 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 to analyze the situations with *full sensing*, i.e., situations in which, for every fluent f_i, 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 $\Pi_2\mathbf{P}$-complete.*

Theorem 9. *For situations with incomplete information and with full sensing, the* 0*-approximation to the planning problem is* $\Pi_2\mathbf{P}$*-complete.*

These results can be represented by the following table:

	exact planning	0-approximation
complete information	**NP**-complete	**NP**-complete
partial information, no sensing	$\Sigma_2\mathbf{P}$-complete	**NP**-complete
limited number of sensing actions	$\Sigma_2\mathbf{P}$-complete	**NP**-complete
unlimited number of sensing actions	**PSPACE**-complete	**PSPACE**-complete
partial information full sensing	$\Pi_2\mathbf{P}$-complete	$\Pi_2\mathbf{P}$-complete

unlimited number
of sensing actions
PSPACE-complete

$\swarrow$ $\searrow$

full sensing — limited sensing
$\Pi_2\mathbf{P}$-complete — $\Sigma_2\mathbf{P}$-complete

$\swarrow$

0-approximation or
complete information
NP-complete

2.6 Auxiliary result: 1-approximation is coNP-complete

In addition to 0-approximation, the authors of [5, 6] considered other types of approximations, including the so-called *1-approximation*. In 1-approximation, partial states are defined in the same manner as for 0-approximation: i.e., as lists of fluents and their negations. However, the result of a (proper) action a on a state s is defined differently: in this new approximation, a fluent expression 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 state $Res_D(a, s)$, we take the set of all fluent expressions 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 0-approximation, computing the next state $Res_D(a,s)$ was a olynomial-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 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 ω-approximation.

3 Proofs

Proof of Theorem 1. First, let us show that for situations with complete information, the planning problem belongs to the class **NP**. Indeed, for a given situation w, checking whether a successful plan exists or not means checking the validity of the formula $\exists u\, P(u,w)$, where $P(u,w)$ stands for "the plan u succeeds for a situation w". To prove that the planning problem belongs to the class **NP**, it is therefore sufficient to prove the following two statements:

- that the quantifier runs only over words u of feasible length, and

- that the property $P(u,w)$ can be checked in polynomial time.

The first statement immediately follows from the fact that in this paper, we are considering only plans of polynomial (feasible) length, i.e., plans u whose length $|u|$ is bounded by a polynomial of the length $|w|$ of the input w: $|u| \le p(|w|)$, where $p(n)$ is a given polynomial. So, the quantifier runs over words of feasible length.

Let us now prove the second statement. Once we have a plan u of feasible length, we can check its successfulness in a situation w as follows:

- we know the initial state s_0;

- take the first action from the action plan u and apply it to the state s_0; as a result, we get the state s_1;
- take the second action from the action plan u and apply it to the state s_1; as a result, we get the state s_2; etc.

At the end, we check whether in the final state, the desired fluent is indeed true. On each step of this construction, the application of an action to a state requires linear time; in total, there are polynomially many steps in this construction. Therefore, this checking indeed requires polynomial time.

So, the planning problem indeed belongs to the class **NP**. Let us show that it is **NP**-complete. To show it, we will prove that the known **NP**-complete problem – the *propositional satisfiability problem* – can be reduced to this problem. In the propositional satisfiability problem, the input is a *propositional formula* F, i.e., any expression which can be obtained from Boolean ("true"–"false") variables $x_1, \ldots, x_n$ by using propositional operations & ("and"), $\vee$ ("or"), and $\neg$ ("not"). The problem is to check whether the given formula F is *satisfiable*, i.e., whether there exist values $x_1, \ldots, x_n$ which make the formula F true. Let us show how, for each propositional formula F, we can design a planning problem whose solvability is equivalent to satisfiability of the original formula F.

To simplify the desired reduction to a planning problem, let us first reformulate the propositional formula F in a more constructive (action-like) way. Namely, when the values $x_1, \ldots, x_n$ are chosen, then for these values, checking the validity of the formula F is straightforward: a computer can check this validity in polynomial (even linear) time. Let us describe, step by step, how the computer will do this checking. In other words, let us *parse* the formula F. Let us denote the intermediate results of this computation by $x_{n+1}, x_{n+2}, \ldots$ For example, if F is the formula $(x_1 \vee x_2)\&(x_1 \vee \neg x_2)$, the a possible parsing of this formula is as follows:

- we start with the values x_1 and x_2;
- then, we compute the first disjunction $x_3 := x_1 \vee x_2$;
- then, we compute the negation $x_4 := \neg x_2$;
- after that, we are ready to compute the second disjunction $x_5 := x_1 \vee x_4$;
- finally, we compute the truth value of the resulting formula as the conjunction of the two disjunctions: $x_6 := x_3 \& x_5$.

In general, we start with the variables $x_1, \ldots, x_n$, and then, for $k = n+1$, $n+2, \ldots$, we compute the value of x_k in one of the three possible ways:

- either as $x_k := x_{f(k)} \& x_{s(k)}$ for some values $f(k) < k$ and $s(k) < k$;
- or as $x_k := x_{f(k)} \vee x_{s(k)}$ for some values $f(k) < k$ and $s(k) < k$;

- or as $x_k := \neg x_{f(k)}$ for some value $f(k) < k$.

Based on this parsing representation of the original propositional formula, we can construct the desired planning situation. Let x_N denote the last value in the parsing construction. In our planning situation, we will have two actions: a and a^-, and $2N+1$ fluents $x_1, \ldots, x_N, s_0, s_1, \ldots, s_N$.

The intended meaning of these fluents and actions is as follows: In our designed plan, in the first n actions, we select the values of the variables $x_1, \ldots, x_n$, and then, in the remaining $N-n$ actions, we simulate the computation of the formula F. The meaning of the fluent s_i is "we are at moment i".

Initially, s_0 is true, and all other fluents are false. The goal of the plan is to make x_N true.

Two groups of rules describe the effects of actions. Rules from the first group describe the selection of the truth values; it also reflects the fact that each action increases time by one:

$$a \text{ causes } x_i \text{ if } s_{i-1};$$

$$a \text{ causes } s_i \text{ if } s_{i-1};$$

$$a \text{ causes } \neg s_{i-1} \text{ if } s_{i-1};$$

$$a^- \text{ causes } \neg x_i \text{ if } s_{i-1};$$

$$a^- \text{ causes } s_i \text{ if } s_{i-1};$$

$$a^- \text{ causes } \neg s_{i-1} \text{ if } s_{i-1}.$$

Here, i takes values from 1 to n.

Rules from the second group describe the computation process. For every k from $n+1$ to N, depending on which operation computes x_k in terms of $x_{f(k)}$ and $x_{s(k)}$, we get the following set of rules:

- if $x_k := x_{f(k)} \& x_{s(k)}$, then we add the following rules:

$$a \text{ causes } x_k \text{ if } s_{k-1}, x_{f(k)}, x_{s(k)};$$

$$a \text{ causes } \neg x_k \text{ if } s_{k-1}, \neg x_{f(k)};$$

$$a \text{ causes } \neg x_k \text{ if } s_{k-1}, \neg x_{s(k)};$$

$$a \text{ causes } s_k \text{ if } s_{k-1};$$

$$a \text{ causes } \neg s_{k-1} \text{ if } s_{k-1}.$$

- if $x_k := x_{f(k)} \vee x_{s(k)}$, then we add the following rules:

$$a \text{ causes } x_k \text{ if } s_{k-1}, x_{f(k)};$$

$$a \text{ causes } x_k \text{ if } s_{k-1}, x_{s(k)};$$

$$a \text{ causes } \neg x_k \text{ if } s_{k-1}, \neg x_{f(k)}, \neg x_{s(k)};$$

$$a \text{ causes } s_k \text{ if } s_{k-1};$$

$$a \text{ causes } \neg s_{k-1} \text{ if } s_{k-1}.$$

- finally, if $x_k := \neg x_{f(k)}$, then we add the following rules:

$$a \text{ causes } x_k \text{ if } s_{k-1}, \neg x_{f(k)};$$

$$a \text{ causes } x_k \text{ if } s_{k-1}, x_{f(k)};$$

$$a \text{ causes } s_k \text{ if } s_{k-1};$$

$$a \text{ causes } \neg s_{k-1} \text{ if } s_{k-1}.$$

At the beginning, s_0 is true, and all other "temporal" variables s_i are false. One can easily check that if we apply any action (a or a^-) to a state in which s_i is true and all other "temporal" variables s_j, $j \neq i$, are false, then in the resulting state, s_{i+1} is true, and all other temporal variables are false. So, by induction, we can prove that all accessible states are like that. If we are in a state in which s_i is true and s_j are false for all $j \neq i$, we will say that we are *at moment of time i*. In these terms any action increases the time by one. Thus, a possible plan can include no more than N actions; hence, the length of any possible plan does not exceed the length of the input data.

Actions performed at moments of time 1 through n select the truth values of the propositional variables $x_1, \ldots, x_n$. One can easily see that on each step $k > n$, the only action we can apply is the action a, and, as a result of this action, we compute the truth value of the auxiliary variable x_k and increase the time by one.

The variable x_N is originally false. The only rules which can make it true require than we have s_{N-1} true; if we apply any action in a state in which s_{N-1} is true, we get a state in which s_N is true. So, the only way for x_N to be true is for s_N to be true as well.

Since each action increases time by one, no matter what sequence of actions we choose, if we have reached s_N this means that we have also computed the truth value x_N of the original formula F. Thus, the only way for x_N to be true is for the original formula F to be true under the chosen Boolean values $x_1, \ldots, x_n$. So, if the above planning problem is solvable, then the propositional formula F is satisfiable. Vice versa, if the formula F is satisfiable, i.e., is true for some propositional values $x_1, \ldots, x_n$, then we can choose these values in our first n actions, and hence, get the solution to our planning problem.

Thus, the solvability of our planning problem is indeed equivalent to the satisfiability of the original formula F. The reduction is proven, and therefore, the planning problem is **NP**-complete.

Proof of Theorem 2. First of all, let us show that for situations with incomplete information and no sensing actions, the planning problem belongs to the class $\Sigma_2\mathbf{P}$. Indeed, incomplete information means that the initial values of some fluents are unknown. For such problems, the existence of a successful action plan means the existence of an action plan u_1 for which, for every set of values u_2 of the unknown fluents, the plan leads to a success. In mathematical terms, the existence of a successful plan can be thus written as a formula $\exists u_1 \forall u_2\, P(u_1, u_2, w)$, where the predicate $P(u_1, u_2, w)$ describes the fact that for the planning problem w and for the values u_2 of initially unknown fluents, the plan u_1 leads to a success. Now, to prove that this problem belongs to the class $\Sigma_2\mathbf{P}$, we must show that the quantifiers run over variables of feasible length, and that the predicate $P(u_1, u_2, w)$ is feasible.

The quantifier u_1 runs over plans and is, therefore, feasible; the quantifier u_2 runs over sets of values of fluents; each set of values is feasible (its length is equal to the number of unknown fluents), so this quantifier is also feasible. Finally, if we know the values u_2 of all the initially unknown fluents, and if we know the sequence of actions u_1, then we can easily check, step-by-step, whether for these values of fluents, the given sequence of action leads to a success (this can be done exactly as in the proof of Theorem 1). Therefore, the predicate $P(u_1, u_2, w)$ is feasible. So, the planning problem indeed belongs to the class $\Sigma_2\mathbf{P}$.

To prove that the planning problem is $\Sigma_2\mathbf{P}$-complete, we will show that we can reduce, to the planning problem, a problem known to be $\Sigma_2\mathbf{P}$-complete: namely, the problem of checking, for a given propositional formula F with the variables $x_1, \ldots, x_m, x_{m+1}, \ldots, x_n$, whether

$$\exists x_1 \ldots \exists x_m \forall x_{m+1} \ldots \forall x_n\, F.$$

The reduction will be similar to the one from Theorem 1, with two exceptions:

- In the planning problem constructed in the proof of Theorem 1, we assumed that initially, all the variables x_i were initially false. In the new reduction, we assume that only the variables $x_1, \ldots, x_m$ are initially false, and that the values of the remaining variables $x_{m+1}, \ldots, x_n$ are initially unknown.

- Correspondingly, rules from the first group (which generate the values x_i) are only constructed for the values $i \leq m$; for i from $m+1$ to n, we have, instead, "dummy" rules which simply increase time by one:

$$a \textsf{ causes } s_i \textsf{ if } s_{i-1};$$

$$a \textsf{ causes } \neg s_{i-1} \textsf{ if } s_{i-1}.$$

Similarly to the proof of Theorem 1, the only way to make x_N true is to go through a sequence of N actions, in first m of which we choose the truth values of the propositional variables $x_1, \ldots, x_m$, and in the last $N - n$ of which we compute the truth value of the original formula F using the selected values of $x_1, \ldots, x_m$, and the original (unknown) values of the propositional variables $x_{m+1}, \ldots, x_n$. Therefore, the existence of a successful action plan is equivalent to the possibility of choosing the values $x_1, \ldots, x_m$ for which, for all possible values of $x_{m+1}, \ldots, x_n$, the formula F is true. In other words, the existence of an action plan is equivalent to the validity of the formula $\exists x_1 \ldots \exists x_m \forall x_{m+1} \ldots \forall x_n\, F$. The reduction is proven, and so the planning problem in indeed $\Sigma_2\mathbf{P}$-complete.

Proof of Theorem 3. In 0-approximation, the existence of a successful action plan is equivalent to $\exists u P(u, w)$. In this approximation, at any given moment of time, the state is described by a finite set of fluents and their negations, and, if we know the previous state and the action, then we can find the next state in linear time. Therefore, in 0-approximation, similarly to the proof of Theorem 1, we can check the successfulness of a given action plan u for a given initial state w in polynomial time. Since the predicate $P(u, w)$ can be checked in polynomial time, and the quantifier $\exists u$ runs over words of polynomial length, the planning problem belongs to the class **NP**.

The fact that it is **NP**-complete follows from the fact that for the particular case of complete information, 0-approximation coincides with the original planning problem, and for complete information, as we have shown in the proof of Theorem 1, the planning problem is indeed **NP**-complete. The theorem is proven.

Proof of Theorem 4. First of all, let us show that if we allow sensing, then for situations with incomplete information, the planning problem belongs to the class **PSPACE**. Indeed, the existence of an action plan of a (feasible) length L can be reformulated as follows: there exists a first action u_1, such that for every possible sensing result u_2 of this first action (if it is a sensing action), there exists a second action u_3, such that for every possible result u_4 of this second action (if it is a sensing action), there exists a third action u_5, etc., such that at the end, we get the desired value of the goal fluent (for all possible values of still un-sensed fluents). In mathematical terms, the existence of a plan can be thus re-written as

$$\exists u_1 \forall u_2 \exists u_3 \forall u_4 \ldots \forall u_k P(u_1, \ldots, u_k, w),$$

where $u_1, \ldots, u_{k-1}$ represent actions and results of sensing actions, and u_k runs over all possible values of un-sensed (unknown) fluents.

In this construction, we have two quantifiers per action in an action plan + one extra quantifier at the end. Therefore, we totally have $k = 2L+1$ quantifiers; since L is feasible (i.e., bounded by a polynomial of the length of the input), the total number $k = 2L + 1$ of quantifiers is feasible too.

Therefore, to prove that this problem belongs to the class **PSPACE**, it is sufficient to show that the predicate $P(u_1, \ldots, u_k, w)$ is feasible, i.e., that if we know $u_1, \ldots, u_k$, and w, then we can check, in polynomial time, whether this predicate is true. Once we know $u_1, \ldots, u_k, w$, it means that we know the initial situation, and we know the values of all the fluents, both sensed (from u_2, u_4, etc.), and un-sensed (from u_k), and that we know the actual sequence of actions (the first action is u_1, the second is u_3, etc.). Since we know the values of all the fluents, and we know the action plan, we can check, in feasible time, whether this particular action plan leads to success in this particular initial complete-information state. Thus, the predicate $P(u_1, \ldots, u_k, w)$ is indeed polynomial-time, and the planning problem indeed belongs to the class **PSPACE**.

To prove that the planning problem is **PSPACE**-complete, we will show that we can reduce, to the planning problem, a problem known to be **PSPACE**-complete: namely, the problem of checking, for a given propositional formula F with the variables $x_1, \ldots, x_m, x_{m+1}, \ldots, x_n$, the validity of the formula

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots F.$$

This reduction will be a modification of the reduction which we used in our proof of Theorem 1. Similarly to that proof, we will start with parsing the formula F; let x_N denote the last value in the parsing construction.

- In addition to two *proper* actions a and a^-, i.e., actions which actually change the state, we have a third action: a *sensing* action d which senses the value of the fluent x_1.
- In addition to $2N+1$ fluents $x_1, \ldots, x_N, s_0, s_1, \ldots, s_N$, we have additional fluents $s_{1.5}, s_{3.5}, \ldots, s_{i.5}, \ldots$ for all odd integers i between 1 and n.

The new fluents represent "intermediate" moments of time:

- the moment 1.5 is intermediate between moments 1 and 2;
- the moment 3.5 is intermediate between moments 3 and 4; etc.

so that

$$1 < 1.5 < 2 < 3 < 3.5 < 4 < 5 < \ldots < n.$$

Similarly to the proof of Theorem 1, the goal of the plan is to make x_N true. Initially:

- s_0 is true;
- all other fluents s_i are false;
- all fluents $x_1, \ldots, x_n$ are unknown; and
- all fluents $x_{n+1}, \ldots, x_N$ are false.

Similarly to the proof of Theorem 1, two groups of rules describe the effects of actions. Rules from the first group describe the selection of the truth values $x_1, \ldots, x_n$; they also reflects the fact that each action moves us to the next moment of time. Rules corresponding to odd-numbered variables x_{2i+1}, $i = 0, 1, \ldots$ (i.e., variables $x_1, x_3, \ldots$) are similar to the ones used in the proof of Theorem 1:

$$a \textsf{ causes } x_{2i+1} \textsf{ if } s_{2i};$$

$$a \text{ causes } s_{2i+1} \text{ if } s_{2i};$$

$$a \text{ causes } \neg s_{2i} \text{ if } s_{2i};$$

$$a^- \text{ causes } \neg x_{2i+1} \text{ if } s_{2i};$$

$$a^- \text{ causes } s_{2i+1} \text{ if } s_{2i};$$

$$a^- \text{ causes } \neg s_{2i} \text{ if } s_{2i}.$$

Here, i takes all integer values from 0 to $\lfloor n/2 \rfloor$ (i.e., all integer values i for which $1 \leq 2i+1 \leq n$).

Rules corresponding to each even-numbered variable x_{2i}, $i = 1, 2, \ldots$, include three steps whose goal is to detect ("sense") the value of this variable by using the sensing action d:

- first, we swap the variable x_{2i} with the variable x_1, thus enabling d to measure the value of what is now x_1 (and what was originally x_{2i});
- then, we actually sense the value of x_1 (which we will be able to later use in selecting further action); and
- finally, we swap back the values x_1 and x_{2i}.

The rules corresponding to the first swap are as follows:

$$a \text{ causes } x_1 \text{ if } x_{2i}, s_{2i-1};$$

$$a \text{ causes } \neg x_1 \text{ if } \neg x_{2i}, s_{2i-1};$$

$$a \text{ causes } x_{2i} \text{ if } x_1, s_{2i-1};$$

$$a \text{ causes } \neg x_{2i} \text{ if } \neg x_1, s_{2i-1};$$

$$a \text{ causes } s_{2i-1.5} \text{ if } s_{2i-1};$$

$$a \text{ causes } \neg s_{2i-1} \text{ if } s_{2i-1}.$$

The rule corresponding to sensing is simple:

$$d \text{ determines } x_1.$$

Finally, the rules corresponding to swap back are as follows:

$$a \text{ causes } x_1 \text{ if } x_{2i}, s_{2i-1.5};$$

$$a \text{ causes } \neg x_1 \text{ if } \neg x_{2i}, s_{2i-1.5};$$

$$a \text{ causes } x_{2i} \text{ if } x_1, s_{2i-1.5};$$

$$a \text{ causes } \neg x_{2i} \text{ if } \neg x_1, s_{2i-1.5};$$

$$a \text{ causes } s_{2i} \text{ if } s_{2i-1.5};$$

$$a \text{ causes } \neg s_{2i-1.5} \text{ if } s_{2i-1.5}.$$

Rules from the second group describe the computation process; these rules are the same as in the proof of Theorem 1.

Let us show that in this situation, the existence of a successful plan is equivalent to the validity of the original propositional formula with quantifiers.

Indeed, if the original propositional formula with quantifiers is true, this means that there exists x_1 such that for every x_2, there exists x_3, etc., for which the formula F is true (i.e., for which x_N is "true"). Here, x_1 is a constant ("true" or "false"), x_3 may depend on x_2, x_5 may depend on x_2 and x_4, etc. In other words, there exists:

- a value x_1;
- a value $x_3(x_2)$ which depends on the previous value x_2;
- a value $x_5(x_2, x_4)$ which may depend on the previous values x_2 and x_4, etc.

for which, for all possible values of $x_2, x_4, \ldots$, the formula $P(x_1, x_2, \ldots)$ is true (this reformulation is called a *skolemization* of the original formula with quantifiers). Therefore, we can use the following action plan to succeed:

- first, at moment 0, we select a or a^- depending on whether the "existing" value of x_1 is "true" or "false";
- then, we use the swap sequence to exchange x_2 and x_1, measure the truth value of x_1, and swap back; as a result, we know the truth value of the variable x_2;
- depending on the sensed value of x_2, we select a or a^- depending on whether $x_3(x_2)$ is true or false;
- then, we apply two swaps and sensing to sense the value of the variable x_4, etc.
- after the moment s_n, we apply the same action (action a) $N-n$ times to compute the truth value $x_N =$"true" of the formula F.

Vice versa, let us assume that for our planning domain, there exists a successful action plan, i.e. an action plan which makes the desired fluent x_N always true. Similarly to the proof of Theorem 1, the only way to make x_N true is to go through a sequence of all moments of time, $s_0, s_1, s_{1.5}, s_2, \ldots, s_n, s_{n+1}, \ldots, s_N$, and the only way to go through this sequence of moments of time is to perform the corresponding actions. In particular, for $x_1, \ldots, x_n$, we must perform all the selecting actions and all the swaps. Of course, there is no necessity to perform the sensing actions, but since performing a sensing action does not change the actual state, we can always add these sensing actions to the action plan without changing the successfulness of this plan. So, without losing generality, we can assume that in the successful action plan, we are sensing the values of all the variables $x_2, x_4, \ldots$ In short, this action plan does the following:

- In the first action, we perform either the action a which leads to x_1, or the action a^- which leads to $\neg x_1$. In other words, in the first action, we select a truth value of the variable x_1.
- Then, we measure x_2, and we select a truth value of the variable x_3. In this selection, we can use our knowledge about x_2; so, the selected value is, in general, a function of x_2: $x_3(x_2)$. (If we do not use x_2, this simply means that we are using a constant function which does not depend on x_2 at all.)
- After that, we measure x_4 and select x_5. In this selection, we can use our knowledge about the values x_2 and x_4, so, in general, the selected value x_5 is a function of x_2 and x_4: $x_5 = x_5(x_2, x_4)$.
- $\ldots$
- After we have selected and sensed the values $x_1, \ldots, x_n$, the resulting actions simply simulate the process of computing the truth value (x_N) of the propositional formula $F(x_1, \ldots, x_n)$.

The success of the action plan means that for all possible values $x_2, x_4, \ldots$, the formula

$$F(x_1, x_2, x_3(x_2), x_4, x_5(x_2, x_4), x_6, \ldots)$$

is true. This means exactly that there exists x_1 such that for every x_2, there exists an x_3, for which, for all x_4, etc., the formula $F(x_1, x_2, x_3, \ldots)$ is true. In other words, the existence of a successful action plan means that the original propositional formula with quantifiers is true.

Since we have already proven the implication in the other direction, we can thus conclude that the existence of a successful action plan is *equivalent* to the truth of the original propositional formula. The reduction is proven, and so the planning problem in indeed **PSPACE**-complete.

Proof of Theorem 5. This result can be proven similarly to the proof of Theorem 4:

- Similarly to that proof, we can show that the 0-approximation to the planning problem belongs to the class **PSPACE**.
- The fact that it is **PSPACE**-complete follows from the observation that in the planning situation described (for reduction purposes) in the proof of Theorem 4, at any given moment of time, our knowledge consists exactly in knowing the values of some fluents, while other fluents can take arbitrary

values. In other words, for this situation, every action plan is also 0-approximate, so the existence of *a* successful action plan for this problem is equivalent to the existence of a successful 0-*approximate* action plan.

The theorem is proven.

Proof of Theorem 6. Let us first show that the planning problem belongs to the class $\Sigma_2\mathbf{P}$. Indeed, the existence of a successful plan can be written as $\exists u_1 \forall u_2\, P(u_1, u_2, w)$, where u_1 is an action plan, and u_2 is the set of initial values of all initially unknown fluents. Here, similarly to the proof of Theorem 4, u_2 runs over words of feasible length and $P(u_1, u_2, w)$ is a feasible predicate. The only difference is with u_1:

- previously (in the proof of Theorem 4), the action plan was simply a sequence of actions, while
- now, an action plan can have some sensing actions inside, and the results of these sensing actions determine the following action.

Each sensing action senses no more than k different fluents. Each fluent can have two different values, so after sensing, we have $\leq 2^k$ different sensing results. So:

- If we have a single sensing action in an action plan, the conditional action plan branches itself into $\leq 2^k$ possible branches (unconditional plans).
- If we have two sensing actions, then each of $\leq 2^k$ branches formed after the first sensing action can, by itself, branch into $\leq 2^k$ sub-branches, making it a total of $\leq 2^k \cdot 2^k = 2^{2k}$ branches.
- We are allowing a total of $\leq k$ sensing actions in each action plan, so we have $\leq 2^k \cdot \ldots \cdot 2^k$ (k times) $= 2^{k^2}$ possible branches.

To describe a conditional action plan, we describe all actions sequences which correspond to different branches. The length of each branch is polynomial (i.e., it is bounded by a polynomial of the length $|w|$ of the input), and the number of branches is limited by a constant (2^{k^2}) which does not depend on the length of the input at all. Therefore, the total length $|u_1|$ of this description u_1 is bounded by a polynomial of $|w|$. So, the first quantifier also runs over words of feasible length. Therefore, the problem indeed belongs to the class $\Sigma_2\mathbf{P}$.

We have already proven (in Theorem 4) that for the particular case of no sensing, the planning problem is $\Sigma_2\mathbf{P}$-complete. Therefore, this more general problem is $\Sigma_2\mathbf{P}$-complete as well. The theorem is proven.

Proof of Theorem 7. This proof is related to the proof of Theorem 5 in the same way as the proof of Theorem 6 was related to the proof of Theorem 4: first, we prove that the 0-approximate planning problem belongs to the class **NP** – by using the same coding u_1 of the conditional plans as in the proof of

Theorem 6, and then we observe that since a particular case (no-sensing) of this problem is **NP**-complete, this general problem is **NP**-complete as well.

Proof of Theorem 8. First of all, let us show that for full sensing, the planning problem belongs to the class $\Pi_2\mathbf{P}$. Indeed, since sensing actions do not change the state of a system, there is no harm in applying them first, and thus, determining the values of all the fluents. For each revealed initial state, we have an unconditional action plan. Thus, the existence of a successful *conditional* action plan for situations with full sensing means that for every initial state u_1, there is an (unconditional) action plan u_2 which leads to a success. In mathematical terms, the existence of a successful plan can be thus written as a formula $\forall u_1 \exists u_2\, P(u_1, u_2, w)$, where the predicate $P(u_1, u_2, w)$ describes the fact that for the planning problem w and for the values u_1 of initially unknown fluents, the plan u_2 leads to a success. Similarly to the proof of Theorem 2, we can prove that the quantifiers run over variables of feasible length, and that the predicate $P(u_1, u_2, w)$ is feasible. Thus, for the case of full sensing, the planning problem indeed belongs to the class $\Pi_2\mathbf{P}$.

To prove that the planning problem is $\Pi_2\mathbf{P}$-complete, we will show that we can reduce, to the planning problem, a problem known to be $\Pi_2\mathbf{P}$-complete: namely, the problem of checking, for a given propositional formula F with the variables $x_1, \ldots, x_m, x_{m+1}, \ldots, x_n$, whether

$$\forall x_1 \ldots \forall x_m \exists x_{m+1} \ldots \exists x_n\, F.$$

The reduction will be similar to the one from Theorem 1, with three exceptions:

- In addition to two proper actions, we also have m *sensing* actions $check_i$, $1 \le i \le m$, which sense the values of the variables $x_1, \ldots, x_m$.
- In the planning problem constructed in the proof of Theorem 1, we assumed that initially, all the variables x_i were initially false. In the new reduction, we assume that only the variables $x_{m+1}, \ldots, x_n$ are initially false, and that the values of the remaining variables $x_1, \ldots, x_m$ are initially unknown.
- Correspondingly, rules from the first group (which generate the values x_i) are only constructed for the values $i > m$; for i from 1 to m, we have, instead, "dummy" rules which simply increase time by one:

  $$a \textsf{ causes } s_i \textsf{ if } s_{i-1};$$

  $$a \textsf{ causes } \neg s_{i-1} \textsf{ if } s_{i-1},$$

  and the "sensing" rules

  $$check_i \textsf{ determines } x_i.$$

Similarly to the proof of Theorem 1, the only way to make x_N true is to go through a sequence of N actions:

- in the first m of these actions, we sense the truth values of the variables $x_1, \ldots, x_m$;
- in the next $n-m$ of these actions, we choose the truth values of the propositional variables $x_{m+1}, \ldots, x_n$; in this choice, we can use the "measured" values of $x_1, \ldots, x_m$;
- finally, in the last $N-n$ actions, we compute the truth value of the original formula F using the "sensed" truth values of the propositional variables $x_1, \ldots, x_m$, and the selected truth values of the propositional variables $x_{m+1}, \ldots, x_n$.

Therefore, the existence of a successful action plan is equivalent to the possibility that for every possible combination of the values $x_1, \ldots, x_m$, we can choose the values $x_{m+1}, \ldots, x_n$ for which the formula F is true. In other words, the existence of an action plan is equivalent to the validity of the formula $\forall x_1 \ldots \forall x_m \exists x_{m+1} \ldots \exists x_n\, F$. The reduction is proven, and so the planning problem in indeed $\Pi_2\mathbf{P}$-complete.

Proof of Theorem 9. We already know, from Theorem 8, that for full sensing, the planning problem is $\Pi_2\mathbf{P}$-complete. To prove that the the existence of a 0-approximate plan is $\Pi_2\mathbf{P}$-complete, it is therefore sufficient to show that for situations with full sensing, the existence of a successful action plan is equivalent to the existence of a 0-approximate action plan.

In one direction this implication is trivial: it is known [5, 6] that a successful 0-approximate action plan is a particular case of a successful plan. Thus, if there exists a successful 0-approximate plan, this means that there exists a successful plan.

Vice versa, let us assume that there exists a successful (conditional) action plan. Since we have a situation with full sensing, we can, in principle, do the following:

- first, we sense all the fluents, thus determining completely the initial state;
- then, we follow the sequence of actions which is recommended by the original conditional plan for this particular initial state.

For complete states, every plan is a 0-approximate plan. Therefore, what we described is a successful 0-approximate plan.

Th equivalence between the existence of a successful plan and the existence of a successful 0-approximate plan is thus proven, and therefore, the 0-approximation to the planning problem is indeed $\Pi_2\mathbf{P}$-complete.

Proof of Theorem 10. First, let us show that this problem belongs to the class **coNP**. Indeed, the fact that f is true is $Res_D(a, s)$ can be reformulated as

$\forall u P(u,w)$, where u runs over all possible states complementing s, and $P(u,w)$ means that the predicate f is true in the result of applying a to the complete state u. Here, the quantifier runs over complete states – i.e., words of feasible length, and the predicate $P(u,w)$ can also be easily checked in polynomial time. Thus, this problem indeed belongs to the class **coNP**.

To prove that this problem is **coNP**-complete, let us reduce, to this problem, a problem known to be **coNP**-complete: namely, the problem of checking whether a given propositional formula F with n propositional variables $x_1,\ldots,x_n$ is a *tautology*, i.e., whether it is true for all possible values of its variables $x_1,\ldots,x_n$. It is known that this problem is **coNP**-complete even if we restrict ourselves to propositional formulas of the special type: namely, to 3-*CNF* formulas, i.e., formulas of the type $C_1 \,\&\, C_2 \,\&\, \ldots \,\&\, C_k$, where each "clause" C_i is of the type $p \vee q \vee r$, with p, q, and r being literals (i.e., propositional variables x_i or their negations).

Let us now show how we can reduce an instance of a CNF-tautology problem to checking whether f holds in $Res_D(a,s)$. Let $C_1 \,\&\, C_2 \,\&\, \ldots \,\&\, C_k$ be a checked formula F with propositional variables $x_1,\ldots,x_n$. Then, we define a planning situation with $n+1$ fluents $f, x_1,\ldots,x_n$. In the initial state s, f is true, and fluents $x_1,\ldots,x_n$ are unknown. We have k rules which describe the result of the action a – one rule for each clause C_j. Namely, for each clause $p \vee q \vee r$, we have a rule

$$a \ \mathsf{causes}\ \neg f \ \mathsf{if} \neg p, \neg q, \neg r.$$

Let us show that f is true in $Res_D(a,s)$ if and only if the original formula F is a tautology. Indeed, initially f was true; the only reason for it to stop being true if for some state u complementing s, we get $\neg f$, i.e., if for some values of the variables $x_1,\ldots,x_n$, for one of the clauses $C_j \equiv p \vee q \vee r$, we have $\neg p \,\&\, \neg q \,\&\, \neg r$. But this conjunction is exactly the negation of the clause, so, in other words, f is not true in $Res_D(a,s)$ if and only if for some values of the variables $x_1,\ldots,x_n$, one of the clauses is false.

Therefore, f *is* true in $Res_D(a,s)$ if and only if for every choice of the variables $x_1,\ldots,x_n$, all clauses C_j are true – which is equivalent to saying that the original formula $C_1 \,\&\, \ldots \,\&\, C_k$ is true. So, f is true in $Res_D(a,s)$ if and only if the original formula is a tautology. The reduction is proven, and so our problem is indeed **coNP**-complete.

Acknowledgments. This work was supported in part by NASA under cooperative agreement NCC5-209, by NSF grant No. DUE-9750858, by United Space Alliance, grant No. NAS 9-20000 (PWO C0C67713A6), by Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518, and by the National Security Agency under Grant No. MDA904-98-1-0564.

References

[1] 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).

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

[3] P. Liberatore, "The complexity of the language $\mathcal{A}$", *Electronic Transactions on Artificial Intelligence*, 1997, Vol. 1, pp. 13–28 (http://www.ep.liu.se/ej/etai/1997/02).

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

[5] T. Son, and C. Baral, "Approximate reasoning about actions in presence of sensing and incomplete information", In: *Proc. of International Logic Programming Symposium (ILPS'97)*, 1997, pp. 387–401.

[6] 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).