Computational Complexity of Planning with Temporal Goals

Chitta Baral
CS&E, Arizona State University
Tempe, AZ 85287-5406, USA
chitta@asu.edu

Vladik Kreinovich
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@cs.utep.edu

Raúl A. Trejo
ITESM Campus Edo. México
Atizapan, México 52926
rtrejo@campus.cem.itesm.mx

Abstract

In the last decade, there has been several studies on the computational complexity of planning. These studies normally assume that the goal of planning is to make a certain fluent true after the sequence of actions. In many real-life planning problems, the goal is represented in a much more complicated temporal form: e.g., in addition to having a desired fluent true at the end, we may want to keep certain fluents true at all times. In this paper, we study the complexity of planning for such temporal goals. We show that for goals expressible in Linear Temporal Logic, planning has the same complexity as for non-temporal goals: it is **NP**-complete; and for goals expressible in a more general Branching Temporal Logic, planning is **PSPACE**-complete.

1 Introduction

In the presence of complete information about the initial situation, a plan – in the sense of classical planning – is a sequence of actions that takes the agent from the initial situation to the state which satisfies a given goal. Traditionally, a goal is described as a fluent which must be true after all the actions. For such goals, the computational complexity of finding a plan has been well-studied [Bylander, 1994; Erol *et al.*, 1995; Liberatore, 1997; Baral *et al.*, 2000]. In the most natural formulation, the problem of finding polynomial length plans is **NP**-complete (for exact definitions of standard complexity terms such as **NP**-completeness, see, e.g., [Papadimitriou, 1994; Baral *et al.*, 2000]).

In many real-life planning problems, the goal is represented in a much more complicated temporal form: e.g., in addition to having a desired fluent true at the end, we may want to keep certain fluents true at all times; for example, we may want to make sure that certain safety constraints are satisfied at all times. *In this paper, we study the complexity of planning for such temporal goals.* There exist two formalisms for describing temporal goals: *Linear Temporal Logic* (LTL) [Bacchus and Kabanza, 1998] in which we are allowed to refer to the *actual* past and future events, and *Branching Temporal Logic* CTL [Niyogi and Sarkar, 2000] in which we are also allowed to refer to events from the *possible* future. In this paper, we will describe the computational complexity of planning in both logics. To the best of our knowledge this has not been done before.

Our complexity analysis will be based on the action description language $\mathcal{A}$ proposed in [Gelfond and Lifschitz, 1993]. The language $\mathcal{A}$ and its variants have made it easier to understand the fundamentals (such as inertia, ramification, qualification, concurrency, sensing, etc.) involved in reasoning about actions and their effects on a world, and we would like to stick to that simplicity principle here. To stick to the main point we consider the simplest action description, and do not consider features such as executability conditions. We now start with a brief description of the language $\mathcal{A}$.

1.1 The 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 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

$$\mathsf{initially}\ f,$$

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

$$a\ \mathsf{causes}\ f\ \mathsf{if}\ f_1, \ldots, f_m,$$

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 execution of an action a, fluent literals $f_1, \ldots, f_m$ were true, and the domain description contains a rule "a causes f if $f_1, \ldots, f_m$", then this rule is *activated*, and after the execution of the action a, f becomes true.
- If for some fluent f_i, no activated rule enables us to conclude that f_i is true or false, this means that the execution of action a does not change the truth of this fluent; therefore, f_i is true in the resulting state if and only if it was 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). The *initial state* s_0 consists of all the fluents f_i for which the corresponding value proposition "initially f_i" is contained in the domain description. (Here we are assuming that we have complete information about the initial situation.) 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 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 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 *defined* 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* $Res_D(p, s)$ of applying a plan p to the initial state s_0 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 property, find a plan for which the resulting trajectory s_0, $s_1 \stackrel{\text{def}}{=} Res_D(a_1, s_0)$, $s_2 \stackrel{\text{def}}{=} Res_D(a_2, s_1)$, etc., satisfies the desired property. In particular, if the goal is to make a certain fluent f true, then the planning problem consists of finding a plan which leads to the state in which f is true.

In addition to the planning problem, it is useful to consider the *plan checking* problem: given a domain, a desired property, and a candidate plan, check whether this action plan satisfies the desired property. It is known that in the presence of complete information about the initial situation, for fluent goals, plan checking is a *tractable* problem – i.e., there exists a polynomial-time algorithm for checking whether a given plan satisfies the given fluent goal [Bylander, 1994; Erol *et al.*, 1995; Liberatore, 1997; Baral *et al.*, 2000].

1.2 Temporal extensions: motivations

Let us give two examples of planning problems explaining why temporal extensions are desirable:

1) If we are planning a flight of an automatic spy miniplane, then the goal is not only to reach the target point (which can be described by the fluent r), but also to avoid detection; more formally, a fluent d ("detected") must remain false all the time.

2) When planning a movement of a robot, we may want to require that not only the robot achieve its goal but also that, whenever the robot strays from the desired trajectory, it should always be possible to bring the robot back to this trajectory (i.e., make the fluent $on\_trajectory$ true) by a single corrective action.

1.3 Linear temporal logic: brief reminder

In Linear Temporal Logic (LTL), in addition to the truth values of a fluent at the current moment of time, we can also refer to its truth values in the past and in the future. For this, LTL has several *operators*. Different authors use different notations for these operators. Since we will also analyze planning in branching time logic described in [Niyogi and Sarkar, 2000] as an extension of LTL, we will use notations from [Niyogi and Sarkar, 2000] for LTL operators.

LTL has four basic *future* operators:

- X (neXt time in the future): $\mathsf{X}p$ is true at a moment time t if p is true at the moment $t + 1$;
- G (Going to be always true): $\mathsf{G}p$ is true at the moment t is p is true at all moments of time $s > t$.
- F (sometime in the Future): $\mathsf{F}p$ is true at the moment t if p is true at some moment $s > t$;
- U (Until): $p\mathsf{U}q$ is true at the moment t if p is true at this moment of time and at all the future moments of time until q becomes true.

Similarly, LTL has four basic *past* operators:

- P (Previously): $\mathsf{P}p$ is true at a moment time t if p is true at the moment $t - 1$;
- H (Has always been): $\mathsf{H}p$ is true at the moment t if p was true at all moments of time $s < t$;
- O (Once or sometime in the past): $\mathsf{O}p$ is true at the moment t if p was true at some moment $s < t$;
- S (Since): $p\mathsf{S}q$ is true at the moment t if p is true at this moment of time and at all the past moments of time since the last time when q was true.

We can combine several such operators: e.g., $\mathsf{X}^3 p \stackrel{\text{def}}{=} \mathsf{XXX}p$ is true at a moment t if p is true at the moment $t + 3$.

In general, an *LTL-goal* is a goal which is obtained from fluents by using LTL operators and propositional connectives & ("and"), $\vee$ ("or"), and $\neg$ ("not").

For example, the objective from our first planning problem can be easily formulated as the following LTL-goal: $S \stackrel{\text{def}}{=} r \,\&\, \neg d \,\&\, \mathsf{H}(\neg d)$.

Comment. Some versions of LTL have additional operators, e.g., we may have *interval* operators in which moments of time $s > t$ or $s < t$ are restricted to a given interval [Bacchus and Kabanza, 1998; Niyogi and Sarkar, 2000].

1.4 Branching temporal logic: brief reminder

In the Branching Temporal Logic CTL [Emerson, 1990; Niyogi and Sarkar, 2000], in addition to LTL operations, we have two additional operators E and A which describe possible futures:

- $\mathsf{E}p$ (Exists path) is true at the state s at the time t if there exists a possible evolution of this state for which p is true at this same moment t.

- $\mathsf{A}p$ (All paths) is true at the state s at the time t if for all possible evolutions of this state, p is true at this same moment t.

For example, the requirement that, no matter what action we apply to the state s, a fluent f will always stay true, can be described as $\mathsf{A}(\mathsf{X}f)$.

Similarly, we can describe in this language the *fast maintainability* requirement from our second planning problem: no matter what action we apply to the state s, if a fluent f stops being true after this action, we can always make the property f true by applying appropriate correcting action.

In CTL, this fast maintainability requirement can be formulated as follows:

- Once we have already reached the next state s', the possibility to get f back by applying a single correcting action means that either f is already true, or there is a path in which f will be true at the next moment of time ($\mathsf{X}f$), i.e., that $f \vee \mathsf{E}(\mathsf{X}f)$.
- So, the fast maintainability requirement means that every possible immediate future state s' satisfies the above property $f \vee \mathsf{E}(\mathsf{X}f)$, i.e., in CTL notations, that

$$\mathsf{A}(\mathsf{X}(f \vee \mathsf{E}(\mathsf{X}f))). \quad (1)$$

Comment. The description of more general temporal logics can be found in [Gabbay et al., 1994].

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 *tractable* 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 tractable algorithms as possible.

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 a (sequential) action plan consists of a sequence of 2^{2^n} actions, then this plan is not tractable. It is therefore reasonable to restrict ourselves to *tractable* plans, i.e., to plans u whose duration $T(u)$ is bounded by a polynomial $p(|w|)$ of the input w.

With this tractability in mind, we can now formulate the above planning problem in precise terms:

- *given:* a polynomial $p(n) \ge n$, a domain description D (i.e., the description of the initial state and of possible consequences of different actions) and a goal statement S (i.e., a statement which we want to be true),
- *determine* whether it is possible to tractably achieve this goal, i.e., whether there exists a tractable-duration plan u (with $T(u) \le p(|D| + |S|)$) which achieves this goal.

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

2.2 Complexity of the planning problem with goals expressible in Linear Temporal Logic

Theorem 1. *For goals expressible in Linear Temporal Logic* (LTL)*, the planning problem is* **NP***-complete.*

Comments.

- Since the planning problem is **NP**-complete even for simple (non-temporal) goals [Bylander, 1994; Erol *et al.*, 1995; Liberatore, 1997; Baral *et al.*, 2000], this result means that allowing temporal goals from LTL does not increase the computational complexity of planning.
- This result is in good accordance with the fact that the decidability problem for linear temporal logic is also **NP**-complete [Emerson, 1990].
- For readers' convenience, all the proofs are placed in the special Proofs section.

The proof of Theorem 1 is based on the following result:

Theorem 2. *For goals expressible in Linear Temporal Logic* (LTL)*, the plan checking problem is tractable.*

We give the proofs of Theorems 1 and 2 for the version of Linear Temporal Logic which only uses eight basic temporal operators. However, as one can easily see from the proofs, these result remains true if we allow more sophisticated temporal operators, e.g., interval operators of the type $F_{[t,s]}$ meaning that the fluent is true in some future moment of time from the interval $[t, s]$ [Bacchus and Kabanza, 1998; Niyogi and Sarkar, 2000].

2.3 Complexity of the planning problem with goals expressible in Branching Temporal Logic

Theorem 3. *For goals expressible in Branching Temporal Logic* CTL*, the planning problem is* **PSPACE***-complete.*

Comment. This result is in good accordance with the fact that the decidability problem for most branching temporal logics is also **PSPACE**-complete [Gabbay et al., 1994].

For the Branching Temporal Logic, not only planning, but even plan checking is difficult:

Theorem 4. *For goals expressible in Branching Temporal Logic* CTL*, the plan checking problem is* **PSPACE***-complete.*

Theorems 3 and 4 mean that allowing temporal goals from CTL can drastically increase the computational complexity of planning. These results, however, are not that negative because most safety and maintainability-type conditions can be expressed in a simple subclass of CTL. Namely, in the maintainability conditions like the one above, we do not need to consider all possible trajectories, it is sufficient to consider trajectories which differ from the actual one by no more than one (or, in general, by no more than k) states. In this case, the planning problem becomes much simpler:

Definition 1. *Let $k > 0$ be a positive integer. We say that an expression in* CTL *is k-limited if this expression remains true when in all operators* E *and* A*, we only allow possible trajectories differ from the actual trajectory in no more than k moments of time.*

For example, the above maintainability statement (1) means that all possible 1-deviations from the actual trajectory are maintainable and therefore, this statement is 1-limited.

Theorem 5. *Let $k > 0$ be an integer. For k-limited goals expressible in Branching Temporal Logic* CTL*, the planning problem is* **NP***-complete.*

Theorem 6. *Let $k > 0$ be an integer. For k-limited goals expressible in Branching Temporal Logic* CTL*, the plan checking problem is tractable.*

2.4 Conclusions

Our first conclusion is that if, instead of traditional goals which only refer to the state of the system at the last moment of time, we allow goals which explicitly mention the actual past and actual future states, the planning problem does not become much more complex: it stays on the same level of complexity hierarchy.

Our second conclusion is that if we allow goals which refer to *potential* future, the planning problem can become drastically more complicated. Thus, we should be very cautious about such more general goals.

3 Proofs

Proof of Theorems 1 and 2. We already know that the planning problem is **NP**-complete even for the simplest possible case of LTL-goals: namely, for goals which are represented simply by fluents [Bylander, 1994; Erol *et al.*, 1995; Liberatore, 1997; Baral *et al.*, 2000]. Therefore, to prove that the general problem of planning under LTL-goals is **NP**-complete, it is sufficient to prove that this general problem belongs to the class **NP**.

Indeed, it is known [Papadimitriou, 1994] that a problem belongs to the class **NP** if the corresponding formula $F(w)$ can be represented as $\exists u P(u,w)$, where $P(u,w)$ is a tractable property, and the quantifier runs over words of tractable length (i.e., of length limited by some given polynomial of the length of the input).

For a given planning 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 the situation w". According to the above definition of the class **NP**, to prove that the planning problem belongs to the class **NP**, it is sufficient to prove the following two statements:

- the quantifier runs only over words u of tractable length, and
- 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 (tractable) duration, i.e., sequential 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 tractable length.

Let us now prove the second statement – that plan checking can be done in polynomial time. (This statement constitutes Theorem 2.) Once we have a plan u of tractable 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 get the values of all the fluents at all moments of time. On each step of this construction, the application of an action to a state requires linear time; in total, there are polynomial number of steps in this construction. Therefore, computing the values of all the fluents at all moments of time indeed requires polynomial time.

Let us now take the desired goal statement S and *parse* it, i.e., describe, step by step, how we get from fluents to this goal statement S.

> For example, for the above spy-plane goal statement $S \equiv r \,\&\, \neg d \,\&\, \mathsf{H}(\neg d)$, parsing leads to the following sequence of intermediate statements: $S_1 := \neg d$, $S_2 := r \,\&\, S_1$, $S_3 := \mathsf{H}S_1$, and finally, $S \equiv S_4 := S_2 \,\&\, S_3$.

The number of the resulting intermediate statements cannot exceed the length of the goal statement; thus, this number is bounded by the length $|S|$ of the goal statement.

Based on the values of all the fluents at all moments of time, we can now sequentially compute the values of all these intermediate statements S_i at all moments of time:

- When a new statement is obtained from one or two previous ones by a logical connective (e.g., in the above example, as $S_4 := S_2 \,\&\, S_3$), then, to compute the value of the new statement at all T moments of time, we need T logical operations.
- Let us now consider the case when a new statement is obtained from one or two previously computed ones by using one temporal operations: e.g., in the above example, as $S_3 := \mathsf{H}S_1$). Then, to compute the truth value of S_3 at each moment of time, we may need to go over all other moments of time. So, to compute S_i for each moment of time t, we need $\le T$ steps. Hence, to compute the truth value of S_i for *all* T moments of time, we need $\le T^2$ steps.

In both cases, for each of $\le |S|$ intermediate statements, we need $\le T^2$ computations. Thus, to compute the truth value of the desired goal statement, we need $\le T^2 \cdot |S|$ computational steps. Since we look for plans for which $T \le p(|D|+|S|)$ for some polynomial $p(n)$, we thus need a polynomial number of steps to check whether the given plan satisfies the given goal.

So, we can check the success of a plan in polynomial time, and thus, the planning problem indeed belongs to the class **NP**. The theorems are proven.

Proof of Theorem 3. This proof follows the same logic as proofs of **PSPACE**-completeness of other planning problems; see, e.g., [Littman, 1997] and [Baral *et al.*, 2000].

By definition, the class **PSPACE** is 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 $P(w)$ 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 tractable property, and all k quantifiers run over words of tractable length (i.e., of length limited by some given polynomial of the length of the input). In view of this result, it is easy to see that for CTL-goals, the planning problem belongs to the class **PSPACE**. Indeed, all the operators of CTL can be described by quantifiers over words of tractable length, namely, either over paths (for operators A and E) or over moments of time (for LTL operators). A plan is also a word of tractable length. Thus, the existence of a plan which satisfies a given CTL-goal can be described by a tractable sequence of quantifiers running over words of tractable length. Thus, for CTL-goals, the planning problem does belong to **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 $\mathcal{F}$ of the type $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots F$. This reduction will be done as follows. Consider the planning problem with two actions a^+ and a^-, and $2n+1$ fluents $x_1, \ldots, x_n$, $t_0, t_1, \ldots, t_n$. These actions and fluents have the following meaning:

- the meaning of t_i is that we are at moment of time i;
- the action a^+, when applied at moment t_{i-1}, makes i-th variable x_i true;
- the action a^-, when applied at moment t_{i-1}, makes i-th variable x_i false.

The corresponding initial conditions are:

- $\mathsf{initially}\ \neg x_i$ (for all i);
- $\mathsf{initially}\ t_0$; $\mathsf{initially}\ \neg t_i$ (for all $i > 0$).

The effect of actions if described by the following rules (effect propositions):

- for $i = 1, 2, \ldots, n$, the rules

$$a^+ \ \mathsf{causes}\ x_i \ \mathsf{if}\ t_{i-1}; \quad a^- \ \mathsf{causes}\ \neg x_i \ \mathsf{if}\ t_{i-1};$$

  describe how we assign values to the variables x_i;

- for $i = 1, 2, \ldots, n$, the rules

$$a^+ \ \mathsf{causes}\ t_i \ \mathsf{if}\ t_{i-1}; \quad a^- \ \mathsf{causes}\ t_i \ \mathsf{if}\ t_{i-1};$$
$$a^+ \ \mathsf{causes}\ \neg t_{i-1} \ \mathsf{if}\ t_{i-1}; \quad a^- \ \mathsf{causes}\ \neg t_{i-1} \ \mathsf{if}\ t_{i-1};$$

  describe the update of the time fluents t_i.

The corresponding goal is designed as follows:

- first, we replace in the above quantified propositional formula $\mathcal{F}$, each existential quantifier $\exists x_i$ by EX, each universal quantifier $\forall x_i$ by AX; let us denote the result of this replacement by F';
- then, we add $\,\&\, t_0$ to the resulting formula F';
- finally, we add P^n in front of the whole thing – creating $\mathsf{P}^n(F' \,\&\, t_0)$.

For example, for a formula $\exists x_1 \forall x_2 F$, this construction leads to the following goal: $\mathsf{P}^2(\mathsf{EX}(\mathsf{AX}(F)) \,\&\, t_0)$. This reduction leads to a linear increase in length, so this reduction is polynomial-time.

To complete the proof, we must show that this is a "valid" reduction, i.e., that the resulting planning problem is solvable if and only if the original quantified propositional formula is true.

To prove this equivalence, let us first remark that, by definition of the operator P ("previous"), the goal formula $\mathsf{P}^n(F' \,\&\, t_0)$ is true at a moment t if and only if the formula $F' \,\&\, t_0$ holds at a moment $t-n$. Since, due to our rules, t_0 is only true at the starting moment of time $t = 0$, the goal can only be true if $t = n$. Thus, to check whether we can achieve the goal, it is sufficient to check whether we can achieve it at the moment n, i.e., after a sequence of n actions. In this case, the validity of the goal is equivalent to validity of the formula F' at the initial moment $t = 0$.

Let us now show that the validity of the formula F' at the moment $t = 0$ is indeed equivalent to the validity of the above quantified propositional formula. We will prove this equivalence by induction over the total number of variables n.

Induction base: For $n = 0$, we have no variables x_i at all, so F is either identically true or identically false. In this case, F' simply coincides with F, so they are, of course, equivalent.

Induction step: Let us assume that we have proven the desired equivalence for all quantified propositional formulas with $n-1$ variables; let us prove it for quantified propositional formulas with n variables.

Indeed, let a quantified propositional formula $\mathcal{F}$ of the above type be given. There are two possibilities for the first variable x_1 of this formula:

- it may be under the existential quantifier $\exists x_1$; or
- it may be under the universal quantifier $\forall x_1$.

1°. In the first case, the formula $\mathcal{F}$ has the form $\exists x_1 \mathcal{G}$, where for each x_1, $\mathcal{G}$ is a quantified propositional formula with $n-1$ variables $x_2, \ldots, x_n$. According to our construction, the CTL formula F' has the form $\mathsf{E}(\mathsf{X}G')$, where G' is the result of applying this same construction to the formula $\mathcal{G}$.

To show that F' is indeed equivalent to $\mathcal{F}$, we will first show that F' implies $\mathcal{F}$, and then that $\mathcal{F}$ implies F'.

1.1°. Let us first show that F' implies $\mathcal{F}$.

Indeed, by definition of the operator E, if the formula $F' \equiv \mathsf{E}(\mathsf{X}G')$ holds at the moment $t = 0$ this means that there exists a path for which, at moment $t = 0$, the formula $\mathsf{X}G'$ is true.

By definition of the operator X ("next"), the fact that the formula $\mathsf{X}G'$ is true at the moment $t = 0$ means that the formula G' is true at the next moment of time $t = 1$.

By the time $t = 1$, we have applied exactly one action which made x_1 either true or false, after which the value of this variable x_1 does not change. Let us select the value x_1 as "true" or "false" depending on which value was selected along this path.

The moment t_1 can be viewed as a starting point for the planning problem corresponding to the remaining formula $\mathcal{G}$. By induction assumption, the validity of G' at this new starting moment is equivalent to the validity of the quantified propositional formula $\mathcal{G}$. Thus, the formula $\mathcal{G}$ is true for this particular x_1, hence the original formula $\mathcal{F} \equiv \exists x_1 \mathcal{G}$ is also true. So, F' indeed implies $\mathcal{F}$.

1.2°. Let us now show that $\mathcal{F}$ implies F'.

Indeed, if $\mathcal{F} \equiv \exists x_1 \mathcal{G}$ is true, this means that there exists a value x_1 for which $\mathcal{G}$ is true. By the induction assumption, this means that for this same x_1, the goal formula G' is also true at the new starting moment $t = 1$. Thus, for any path which starts with selecting this x_1, the formula $\mathsf{X}G'$ is true at the previous moment $t = 0$. Since this formula is true for *some* path, by definition of the operator E, it means that the formula $\mathsf{E}(\mathsf{X}G')$ is true at the moment $t = 0$, and this formula is exactly F'.

Thus, $\mathcal{F}$ does imply F', and hence $\mathcal{F}$ and F' are equivalent.

2°. The second case, when x_1 is under the universal quantifier $\forall x_1$, can be handled similarly.

The induction step is proven, and thus, by induction, the equivalence holds for all n.

Thus, the reduction is valid, and the corresponding planning problem is indeed **PSPACE**-complete. Q.E.D.

Proof of Theorem 4. Similarly to the proof of Theorem 3, we can show that plan checking belongs to the class **PSPACE**, so all we need to prove is the desired reduction. From the proof of Theorem 3, one can see that the exact same reduction will work here as well, because in this reduction, the equivalence between $\mathcal{F}$ and F' did not depend on any action plan at all. The equivalence used in the proof of Theorem 3 is based on the analysis of *possible* trajectories and does not use the actual trajectory at all.

Thus, we can pick any action plan (e.g., a sequence consisting of n actions a^+), and the desired equivalence will still hold. Q.E.D.

Proof of Theorems 5 and 6. For trajectories of duration T, with A possible actions at each moment of time, there are no more than $T \cdot A$ possible trajectories differing in one state, no more than $(T \cdot A)^2$ trajectories differing in two states, etc. In general, whatever number k we fix, there is only a polynomial number ($\leq (T \cdot A)^k$) of possible trajectories which differ from the actual one in $\leq k$ places.

Therefore, for fixed k, we can explicitly describe the new operators E and A by enumerating all such possible trajectories. Thus, similarly to the proof of Theorems 1 and 2, we can conclude that for k-planning, plan checking is tractable and the corresponding planning problem is **NP**-complete.

Acknowledgments

This work was supported by NASA (grant NCC5-209 and a Research on Intelligent Systems grant), by the AFOSR grant F49620-00-1-0365, by the grant W-00016 from the U.S.-Czech Science and Technology Joint Fund, and by the NSF grants IRI 9501577, 0070463, and 9710940 Mexico/Conacyt.

The authors are thankful to the anonymous referees for valuable comments.

References

[Bacchus and Kabanza, 1998] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

[Baral *et al.*, 2000] Chitta Baral, Raúl Trejo, and Vladik Kreinovich. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122:241–267, 2000.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:161–204, 1994.

[Emerson, 1990] E. Allen Emerson. Temporal and modal logics. In: Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science*, Vol. B, pages 995–1072, MIT Press, Cambridge, Massachusetts, 1990.

[Erol *et al.*, 1995] Kuthulan Erol, Dana S. Nau, and V.S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1-2):75–88, 1995.

[Gabbay et al., 1994] Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press, New York, 1994.

[Gelfond and Lifschitz, 1993] Michael Gelfond and Vladimir Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

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

[Littman, 1997] Michael L. Littman. Probabilistic propositional planning: representations and complexity. In *AAAI 97*, pages 748–754, 1997.

[Niyogi and Sarkar, 2000] Rajdeep Niyogi and Sudeshna Sarkar. Logical specification of goals. In *International Conference on Information Technology ICIT'2000*, Bhubaneswar, India, December 21-23, 2000. Tata McGraw-Hill.

[Papadimitriou, 1994] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994.