Complexity and approximation studies of finding polynomially bounded length plans for temporal goals

Chitta Baral †, Vladik Kreinovich ‡, Sudeshna Sarkar ♯,
Nam Tran †, Raul Trejo ♮ and Xin Zhang †

† Department of Computer Science and Engineering,
Arizona State University,
Tempe, AZ 85233, USA.
{*chitta,nhtran,xin.zhang*}*@asu.edu*

‡ Department of Computer Science,
University of Texas at El Paso,
El Paso, TX 79968, USA.
vladik@cs.utep.edu

♯ Department of Computer Science & Engineering,
Indian Institute of Technology,
Kharagpur, India 721302.
sudeshna@cse.iitkgp.ernet.in

♮ ITESM Campus Edo. México,
Atizapan, México 52926
rtrejo@campus.cem.itesm.mx

May 31, 2003

Abstract

In this paper[1] we consider the problem of planning with temporal goals, focussing on polynomially bounded length plans. Past results about complexity of planning are mostly about finding plans that take the world to one of several desired states, often described using a goal formula. We first consider goals expressed using linear temporal logic and analyze the complexity of planning with respect to such goals for both when the states in the trajectory are complete states, and when they are incomplete states. For the later case we also develop a notion of approximate planning and show its complexity to be lower. We also show that this notion of approximate planning is sound. We then consider goals that also have a knowledge component, and refer to such goals as knowledge temporal goals. We analyze the complexity of planning with respect to such goals, propose a notion of approximate planning which is sound and also analyze the complexity of such planning. Finally, we present several goals that can not be adequately expressed using linear temporal logics. To specify these goals, we propose the use of branching time temporal logics such as CTL and CTL*, and define what it means for a plan to satisfy such a goal. We then analyze the complexity of planning with such goals and identify a variant of such goals which leads to a lower complexity of planning.

[1]This paper is a substantial extension of an IJCAI'01 paper by Baral, Trejo and Kreinovich titled "Computational Complexity of Planning with Temporal Goals", with three additional co-authors. About half the results in this version are new with respect to the IJCAI'01 version.

1 Introduction and Motivation

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 by a fluent formula which must be true in the state reached after executing the plan. For such goals, the computational complexity of finding a plan has been well-studied [4, 7, 11, 2]. In the most natural formulation, the problem of finding a plan whose length is bounded by a given polynomial is **NP**-complete. (We give the definitions of standard complexity terms such as **NP**-completeness in a later section.)

In many real-life planning problems, often the goal is much more than just reaching one of a set of desired states. It may involve putting restrictions on the path such as making sure certain fluents are true throughout the path, or the truth value of certain fluents revert back [20] – after the execution of the plan – to their truth value in the initial state. In [1] use of *Linear Temporal Logics* (LTLs) to express such goals is proposed. In this paper our *first goal* is to study the complexity of *polynomial length* planning with such goals when the planning domain is *specified*[2] *in a high-level language* such as STRIPS or $\mathcal{A}$ [8] and the initial state is completely known.

We then consider the case when there is incomplete information about the initial state, usually leading to incomplete knowledge about the states in the trajectories. We analyze the complexity of planning with respect to LTL goals for such cases. Our analysis shows the complexity to be higher than NP-complete. To counter this we develop a notion of approximate planning with respect to LTL goals. We show that our notion is sound and has a lower complexity. (We consider finding sound approximate notions with lower complexity for planning problems with complexity higher than NP-complete to be important and follow that methodology throughout the paper.)

When dealing with incompleteness, especially in presence of sensing actions, often it is useful to have knowledge goals. Sometimes we need to have both knowledge aspects and temporal aspects in a goal. For example, one may wish to have a goal of determining the value of fluent f but without changing its value in the process. The first part is concerned with the 'knowledge' aspect, while the second part is a restriction on the trajectory and needs temporal operators for its specification. We propose a simple logic that incorporates both temporal and knowledge modalities and analyze the complexity of planning with respect to such goals. Here also, we consider an approximate notion; show it to be sound and analyze its complexity.

Although linear temporal logics (LTLs) can specify certain restrictions on the trajectory corresponding to a plan, certain goal specifications are beyond the expressibility of LTLs. This include cases where an agent is interested in states that are not directly in its planned trajectory but are reachable from states in the trajectory. For example, one may want to specify the goal of going from location A to location B such that for each intermediate location there is a gas station within two steps. In this case the gas station does not have to be in the path (trajectory) taken from A to B. Such a goal can not be expressed in LTLs and needs branching time logics. The *next* goal of our paper is to explore the use of branching time logics such as CTL and CTL* in expressing planning goals beyond the capability of LTLs. In particular, we give several example goals and their representations in CTL* and CTL; and precisely define what it means for a plan to satisfy a goal in CTL and CTL*. The use of CTL for expressing goals in planning was first proposed in [13].

[2] In [5] planning with LTL goals is studied with respect to *arbitrary length* plans and with the planning domain *specified as an automata*. There is no direct correlation between these two type of results.

We then analyze the complexity of polynomial length planning with respect to CTL and CTL* goals. We also identify a variant of CTL and CTL* with respect to which polynomial length planning belongs to a lower complexity class.

Our complexity analysis is based on the action description language $\mathcal{A}$ proposed in [8]. 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 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.

The rest of the paper is organized as follows. In Section 2 we present several background materials. In particular in Section 2.1 we give a brief description of the language $\mathcal{A}$; in Section 2.2 we present syntax and semantics of LTL and define what it means for a plan to satisfy an LTL goal; in Section 2.3 we recall useful complexity notions; and in Section 2.4 we recall useful complexity of planning notions. In Section 3 we present complexity analysis of planning with LTL goals. We start with (Section 3.1) with the case where the initial state is completely known. In Section 3.2 we consider the conformant planning when the initial state is incompletely known, and in Sections 3.3-3.6 we discuss an approximate and sound notion of planning for such a case and analyze its complexity. In Section 4 we present the notion of temporal-knowledge formulas, analyze the complexity of planning with goals represented as such formulas, and also study an approximate notion. In Section 5 we discuss the use of CTL* and CTL in expressing goals, define what it means for a plan to satisfy a CTL or CTL* goal, and give several examples of planning goals in CTL and CTL*. We then present complexity results about planning and plan checking with respect to CTL and CTL* goals, and consider a variant of CTL and CTL* goals with a lower complexity. Finally in Section 6 we conclude.

2 Background

2.1 The action description language $\mathcal{A}$

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 $\{f_1, f_3\}$. Intuitively, a state $\{f_1, f_3\}$ means that in that 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 $\neg f_i$. We assume that we have complete knowledge about the initial state.

Execution of actions may change the state. In the language there is a finite set of *actions*. The effect of each action a is specified 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 "$\mathsf{initially}\ f$" (which describe the initial state), and a finite set of *effect propositions* of the type "$a\ \mathsf{causes}\ f\ \mathsf{if}\ f_1, \ldots, f_m$" (which describe results of actions). Assuming that we have complete information about the initial situation, which is our assumption here, the *initial state* s_0 consists of all the fluents f_i for which the corresponding value proposition "$\mathsf{initially}\ f_i$" is in the domain description. We say that a fluent f_i *holds* in s if $f_i \in s$; otherwise, we say that $\neg f_i$ holds in s. The *transition function* $\Phi_D(a, s)$ which describes the effect of an action a on a state s is defined as follows:

- we say that an effect proposition "$a\ \mathsf{causes}\ f\ \mathsf{if}\ f_1, \ldots, f_m$" 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 "$a\ \mathsf{causes}\ f_i\ \mathsf{if}\ f_1, \ldots, f_m$" is activated in s;
- similarly,
  we define $V_D^-(a, s)$ as the set of all fluents f_i for which a rule "$a\ \mathsf{causes}\ \neg f_i\ \mathsf{if}\ f_1, \ldots, f_m$" 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 $\Phi_D(a, s) = (s \cup V_D^+(a, s)) \setminus V_D^-(a, s)$.

When the domain description D is clear from the context we just write Φ instead of Φ_D.

A *plan* p is defined as a sequence of actions $[a_1, \ldots, a_m]$. The *result* $\Phi_D(p, s)$ of applying a plan p to the initial state s_0 is defined as

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

The *planning problem* is: given a domain D and a desired property, find a plan for which the resulting plan $s_0, s_1 \stackrel{\text{def}}{=} \Phi_D(a_1, s_0), s_2 \stackrel{\text{def}}{=} \Phi_D(a_2, s_1)$, 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 candidate plan satisfies the desired property. It is known that in the presence of complete information about the initial situation and deterministic actions, for fluent goals (i.e., goals requiring that a set of fluents be true in the final state), plan checking is a *polynomial* problem – i.e., there exists a polynomial-time algorithm for checking whether a given plan satisfies the given fluent goal [4, 7, 11, 2].

2.2 Representing planning goals using Linear Temporal Logic

In most planning systems the goals – represented by a logical formula – describe a set of finals states that an agent may want to get into. The plans then involve a sequence of actions that takes the world from a

given initial state to one of the states specified by the goal. Temporal logics play a role when the goal is not just to get to one of a set of a given states but also involves conditions on what are acceptable ways to get there and what are not. In the past [1] it has been suggested that Linear Temporal Logics (LTLs) be used to specify certain kind of goals. LTLs are modal logics with modal operators either referring to the future or to the past. The future operators in LTL are: *next* (denoted by $\bigcirc$), *always* (denoted by $\Box$), *eventually* (denoted by $\Diamond$), and *until* (denoted by U). The past operators in LTL are: *previously*, *always in the past, sometime in the past*, and *since*. In this paper our focus will be on the future operators as planning for temporal goals with only future operators can be easily done by forward search methods. Syntactically an LTL formula is defined as follows:

$$
\begin{aligned}
\langle LTL_formula \rangle ::= \quad & \langle propositional_formula \rangle | \\
& \neg \langle LTL_formula \rangle | \\
& \langle LTL_formula \rangle \wedge \langle LTL_formula \rangle | \\
& \langle LTL_formula \rangle \vee \langle LTL_formula \rangle | \\
& \bigcirc \langle LTL_formula \rangle | \\
& \Box \langle LTL_formula \rangle | \\
& \Diamond \langle LTL_formula \rangle | \\
& \langle LTL_formula \rangle \ \mathsf{U} \ \langle LTL_formula \rangle
\end{aligned}
$$

The truth of LTL formulas are usually defined with respect to an infinite sequence of states, often referred to as a *trajectory*, and a reference state. Intuitively, an LTL formula $\Box p$ is true with respect to a trajectory σ consisting of $s_0, s_1, \ldots$, and a reference state s_j if for all $i \geq j$, p is true in s_i. We now give a formal definition of the truth of LTL formulas consisting of future operators [1]. In the following p denotes a propositional formula, s_i's are states, σ is the trajectory $s_0, s_1, \ldots$, and f_i's denote LTL formulas (with only future operators).

- $(s_j, \sigma) \models p$ iff p is true in s_j.
- $(s_j, \sigma) \models \neg f$ iff $(s_j, \sigma) \not\models f$
- $(s_j, \sigma) \models f_1 \wedge f_2$ iff $(s_j, \sigma) \models f_1$ and $(s_j, \sigma) \models f_2$.
- $(s_j, \sigma) \models f_1 \vee f_2$ iff $(s_j, \sigma) \models f_1$ or $(s_j, \sigma) \models f_2$.
- $(s_j, \sigma) \models \bigcirc f$ iff $(s_{j+1}, \sigma) \models f$
- $(s_j, \sigma) \models \Box f$ iff $(s_k, \sigma) \models f$, for all $k \geq j$.
- $(s_j, \sigma) \models \Diamond f$ iff $(s_k, \sigma) \models f$, for some $k \geq j$.
- $(s_j, \sigma) \models f_1 \ \mathsf{U} \ f_2$ iff there exists $k \geq j$ such that $(s_k, \sigma) \models f_2$ and for all i, $j \leq i < k$, $(s_i, \sigma) \models f_1$.

Since truth of LTL formulas are defined with respect to a reference state and a trajectory made up of an infinite sequence of states, to define the correctness of a plan with respect to an initial state and an LTL goal, we need to identify a trajectory that corresponds to the initial state and the plan. We define it as follows:

Let s be a state designated as the initial state, let $a_1, \ldots, a_n$ be a sequence of deterministic actions whose effects are described by a domain description. The trajectory corresponding to s and $a_1, \ldots, a_n$ is the sequence of states $s_0, s_1, \ldots,$ that satisfies the following conditions:

- $s = s_0$,
- $s_{i+1} = \Phi(a_{i+1}, s_i)$, for $0 \leq i \leq n-1$, and
- $s_{j+1} = s_j$, for $j \geq n$.

We then say that the sequence of actions $a_1, \ldots, a_n$ is a plan from the initial state s for the goal f, if $(s_0, \sigma) \models f$, where σ is the trajectory corresponding to s and $a_1, \ldots, a_n$.

One nuance of the above definition is that a simple goal of reaching a state where f is true can not be expressed by just f, but now needs to be expressed by the temporal formula $\Diamond\Box f$. This is because our reference point in the definition of a plan is the initial state. On the other hand if we were to use an LTL with only past operators, then the specification can refer to the states before the reference state, and by having the reference state as the current state during forward planning, the goal of reaching a state where f is true can be expressed by just f.

Another nuance of our definition is that although a propositional formula is also an LTL formula, a goal represented as a propositional formula is in general not very useful as (i) if the formula is true in the initial state then the empty sequence of actions is a plan, and (ii) if the formula is not true in the initial state then there are no plans.

We prefer the usage of future operators because of its use in earlier work on planning with temporal goals [1], because if we use the initial state as the reference point then it remains the same as the plan is expanded in the forward direction, and because we find it more intuitive for expressing many other kind of temporal goals. We now list some temporal goals and their representation using an LTL with only future operators.

1. If we are planning a flight of an automatic spy mini-plane, then the goal is not only to *reach* the target point, but also to avoid detection; i.e., to have the fluent $detected$ false all the time.

   The above can be expressed as: $\Diamond\Box reached \ \wedge \ \Box\neg detected$

2. The goal "until the destination is reached the robot keeps its front clear of obstacles" can be expressed as:

   $clear \cup reached$

3. The goal "until the destination is reached the robot maintains its front clear of obstacles" can be expressed as:

   $\Diamond clear \cup reached$

4. The goal "reach the destination but while doing it if a closed door is opened then it must be immediately closed" can be expressed as:

   $\Diamond\Box reached \ \wedge \ \Box(closed \wedge \bigcirc\neg closed \Rightarrow \bigcirc\bigcirc closed)$.

5. The goal "reach the destination but while doing it if a closed door is opened then it must be eventually closed" can be expressed as:

   $\Diamond\Box reached \ \wedge \ \Box(closed \wedge \bigcirc\neg closed \Rightarrow \Diamond\Box closed)$.

6. The goal "reach the destination but while doing it closed doors must not be opened" can be expressed as:

   $\Diamond\Box reached \wedge \Box(closed \Rightarrow \bigcirc closed)$.

7. When dealing with plans that are not finite, which happens when the agent is continually interacting with the environment, then one way to express that the fluent f be maintained is through the LTL formula: $\Box\Diamond f$.

   This is similar to the notion of stability and stabilizability in discrete event dynamic systems [14].

Additional examples of use of LTL in specifying planning goals are given in the literature [1, 13].

2.3 Useful complexity notions

In this section we briefly review the various complexity notions that we will use in the remaining of the paper. 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 $S = \{w \mid P(w)\}$).

Definition 1 The basic complexity classes

- A decision problem is said to belong to the class **P** if there exists a deterministic Turing machine (DTM) that takes polynomial time in solving this problem.
- A decision problem is said to belong to the class **NP** if there exists a non-deterministic Turing machine (NDTM) that takes polynomial time in solving this problem.
- A decision problem is said to belong to the class **coNP** if the complement of the problem is in **NP**.
- A decision problem is said to belong to the class **PSPACE** if there exists a DTM that takes polynomial space in solving this problem.
- For any deterministic or non-deterministic complexity class C, the class C^A is defined to be the class of all languages decided by machines of the same sort and time bound as in C, except that the machine now has an oracle A.
- The Polynomial hierarchy is defined as follows:
  - $\Sigma_0\mathbf{P} = \Pi_0\mathbf{P} = \mathbf{P}$
  - $\Sigma_{i+1}\mathbf{P} = \mathbf{NP}^{\Sigma_i\mathbf{P}}$
  - $\Pi_{i+1}\mathbf{P} = \mathbf{coNP}^{\Sigma_i\mathbf{P}}$ $\Box$

In this paper we will use the following alternative characterization of the polynomial hierarchy in our proofs.

- A problem belongs to the class **NP** if the formula $w \in S$ (equivalently, $P(w)$) can be represented as $\exists u P(u, w)$, where $P(u, w)$ is a polynomial property, and the quantifier runs over words of polynomial length (i.e., of length limited by some given polynomial of the length of the input). The class **NP** is also denoted by Σ_1**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 formula $w \in S$ (equivalently, $P(w)$) can be represented as $\forall u P(u, w)$, where $P(u, w)$ is a polynomial property, and the quantifier runs over words of polynomial length (i.e., of length limited by some given polynomial of the length of the input). The class **coNP** is also denoted by Π_1**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 Σ_k**P** if the 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 polynomial property, and all k quantifiers run over words of polynomial 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 Π_k**P** if the 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 polynomial property, and all k quantifiers run over words of polynomial length (i.e., of length limited by some given polynomial of the length of the input).

- All these classes Σ_k**P** and Π_k**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., [15]) that this class can be equivalently reformulated as a class of problems for which the 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 polynomial property, and all k quantifiers run over words of polynomial 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, any other general problem from this class can be reduced to it by a polynomial-time reduction.

2.4 Complexity of planning notions

We are now ready to describe the complexity of planning notions. The planning problem of our interest in this paper is as follows:

- *given* a domain description (i.e., the description of the initial state and of possible consequences of different actions) and a goal in a temporal logic,

- *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 *polynomial* algorithm, i.e., by an algorithm $\mathcal{U}$ whose computational time $t_{\mathcal{U}}(w)$ on each input w is bounded by a polynomial

$p(|w|)$ of the length $|w|$ of the input w: $t_{\mathcal{U}}(x) \leq p(|w|)$ (this length can be measured bit-wise or symbol-wise). Recall that 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 polynomial 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 polynomial. It is therefore reasonable to restrict ourselves to *polynomial* plans, i.e., to plans u whose execution time (or 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 $p(n) \geq n$, 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 polynomial-duration plan u (with $T(u) \leq p(|D| + |S|)$) which achieves this goal.

The main goal of this paper is to analyzing the *computational complexity* of this planning problem for goals expressed in various temporal languages, and for various assumptions about the initial state.

3 Complexity and approximation studies of planning with LTL goals

In this section we present our complexity results about planning with respect to goals specified in LTL. We start with the complexity of plan checking and planning with respect to complete initial states and LTL goals.

3.1 Complexity of planning for the complete initial state case and LTL goals

Theorem 3.1 For goals expressible in Linear Temporal Logic (LTL), the plan checking problem is polynomial. □

Proof of Theorem 3.1 Given a plan u of polynomial length, we can check its correctness in a situation w as follows:

- we know the initial state s_0;
- we 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;
- we 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 consider how to build it up step by step from its sub-formulas starting from the fluents.

> For example, for the spy-plane goal statement $S \equiv \Diamond\Box reached \wedge \Box\neg detected$ in Section 2.2, s made up of the following sequence of intermediate statements: $S_1 := reached$, $S_2 = \Box S_1$, $S_3 = \Diamond S_2$, $S_4 = \neg detected$, $S_5 = \Box S_4$, $S_6 = S_3 \wedge S_5$.

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 $S_6 := S_3 \wedge S_5$), 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 operation: e.g., in the above example, as $S_3 := \Diamond S_2$). 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 $\leq T$ steps. Hence, to compute the truth value of S_i for *all* T moments of time, we need $\leq T^2$ steps.

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

Theorem 3.2 For goals expressible in Linear Temporal Logic (LTL), the planning problem is **NP**-complete. □

Proof of Theorem 3.2 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 [4, 7, 11, 2]. 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 [15] 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 polynomially verifiable property, and the quantifier runs over words of polynomial 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 polynomial 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 duration, i.e., sequential plans u whose length $|u|$ is bounded by a given polynomial of the length $|w|$ of the input w: $|u| \leq p(|w|)$, where $p(n)$ is a given polynomial. So, the quantifier runs over words of polynomial length.

Since from Theorem 3.1 we can check the success of a plan in polynomial time, and thus, the planning problem indeed belongs to the class **NP**. The theorem is proven. □

Since the planning problem is **NP**-complete even for simple (non-temporal) goals [4, 7, 11, 2], the above result means that allowing temporal goals from LTL does not increase the computational complexity of planning.

We gave the proofs of Theorems 3.1 and 3.2 for the version of Linear Temporal Logic which only uses future temporal operators. However, as one can easily see from the proofs, these result remains true if we allow past temporal operators or more sophisticated temporal operators, e.g., temporal operators of the type $\Diamond_{[t,s]}$ from [1] which are defined on timed sequences of states. There, a timed sequence of states M is defined as consisting of a sequence of states $s_0, \ldots,$ together with an associated timing function $\mathcal{T}$ that maps states to a point in the non-negative real line such that for all i, $\mathcal{T}(s_i) \leq \mathcal{T}(s_{i+1})$, and for all real numbers r there exists an i such that $\mathcal{T}(s_i) > r$. The entailment $(s_j, M) \models \Diamond_{[t,s]} f$ is then said to hold if there exists an s_k, such that $t \leq \mathcal{T}(s_k) \leq s$, and f is true in s_k. Using $\Diamond_{[t,s]}$, the goal that f must be satisfied in the future during the time interval 10 to 36 is expressed as $\Diamond_{[10,36]} f$.

3.2 Planning w.r.t incomplete initial states and LTL goals: conformant plans

As we mentioned earlier past research on planning with temporal goals has assumed the initial state (as well as the states in the trajectory) to be completely known. Now let us consider the case when we have incomplete information about the initial state. In that case the straightforward approach is to consider all possible initial states that agree with our knowledge about the initial state. Then conformant planning involves searching for a sequence of actions that is a plan with respect to each of these complete initial states and the given LTL goal. We will now formally define this notion and present the complexity results of such conformant planning.

An incomplete state (also referred to as an approximate state or *a-state*) is represented by a consistent set of fluent literals. An *a-state* s is said to be complete if for every fluent f either f or $\neg f$ is in s.

Definition 2 (Extension of an incomplete state) Let s be an incomplete state. We say a state $s' \supseteq s$ is a complete extension of s if s' is complete. □

Definition 3 (Conformant plan) Let D be a domain description, s be an (incomplete) initial state and G be an LTL goal. A sequence of actions $a_1, \ldots, a_n$ is said to be a conformant plan with respect to a (possibly incomplete) initial state s' and goal G, if for every complete extension s of s', $(s, \sigma) \models f$, where σ is the trajectory corresponding to s and $a_1, \ldots, a_n$. □

In [2] it is shown that (conformant) planning with respect to incomplete initial states and goals that are propositional formulas is $\Sigma_2 P$-complete. We now show that even if the goal is an LTL formula the complexity of planning is still $\Sigma_2 P$-complete.

Theorem 3.3 For situations of conformant planning with incomplete initial state, and LTL goals the planning problem is $\Sigma_2 P$-complete. □

Proof: From [2] we already know that the planning problem is Σ_2P-complete for incomplete initial states and for the simplest possible case of LTL-goals: namely, for goals which are represented simply by fluent formulas. Therefore, to prove that the general problem of planning under incomplete initial state and LTL goals is Σ_2P-complete, it is sufficient to prove that this general problem belongs to the class Σ_2P.

Since an incomplete initial state means that the initial values of some fluents are unknown, for such problems, the existence of a successful plan means the existence of a plan u_1 for which, for every set of values u_2 of the unknown fluents, the plan leads to a success. I.e., 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, in order to prove that the problem belongs to the class $\Sigma_2 P$, we need to show that the quantifiers run over variables of polynomial length, and that the predicate $P(u_1, u_2, w)$ is polynomially verifiable. Since the quantifier u_1 runs over plans, it is of polynomial length. The quantifier u_2 runs over sets of values of fluents, and each set of values is of polynomial size (the length is equal to the number of unknown fluents); therefore u_2 is also of polynomial size. 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 as described in the proof of Theorem 2 in [2] we can check step-by-step whether for these values of fluents, the given sequence of actions leads to success. Therefore, the predicate $P(u_1, u_2, w)$ is polynomially verifiable. So the planning problem with respect to LTL goals and incomplete initial states indeed belongs to the class $\Sigma_2 P$. □

3.3 Approximate planning with LTL goals and incomplete initial state

In this section our goal is to develop a notion of approximate planning which can find correct plans – with respect to temporal goals and incomplete initial state, and which belongs to (as we will show) a lower complexity class (of NP-complete) than $\Sigma_2 P$-complete of the previous section. The price we pay is that it may miss finding some plans.

The main initial steps in this endeavor is to: (i) use the notion of a-states and define a sound transition between a-states due to actions, and (ii) define what it means for a temporal formula to be $true$, $false$ and $unknown$ with respect to a trajectory of incomplete states, and show that this definition is sound. For (i) we use the notion of 0-approximation from [19], and *the formulation of (ii) is a novel contribution of this paper*. We now define the transition between a-states due to actions corresponding to the 0-approximation.

Definition 4 (0-transition function) *[19]* The *transition function* Φ_D^0 corresponding to the actions descriptions of a domain description D, which describes the effect of actions on a-states is defined as follows:

- we say that an effect proposition "a **causes** f **if** $f_1, \ldots, f_m$" is *activated* in an a-state s if all m fluent literals f_1,...,f_m hold in s;
- we say that an effect proposition "a **causes** f **if** $f_1, \ldots, f_m$" is *possibly activated* in an a-state s if all m fluent literals f_1,...,f_m possibly hold in s (i.e., are either true or unknown in s);
- for an action a, and a-state s, we define $V_D(a, s)$ as the set of all fluent literals f for which a rule "a **causes** f **if** $f_1, \ldots, f_m$" is activated in s;
- for an action a, and a-state s, we define $V'_D(a, s)$ as the set of all fluent literals f for which a rule "a **causes** f **if** $f_1, \ldots, f_m$" is possibly activated in s;
- for an action a, and a-state s, we say $\Phi^0_D(a, s)$ is undefined if $V'_D(a, s)$ has both f and $\neg f$ for some fluent f. If $\Phi^0_D(a, s)$ is not undefined then we define $\Phi^0_D(a, s)$ as:

  $\{f \mid (f \in s \text{ or } f \in V_D(a, s)) \text{ and } \bar{f} \notin V'_D(a, s)\}$ □

Proposition 1 *[19]* Φ^0_D is sound with respect to Φ_D; I.e., for any a-state s and action a,

- If $\Phi^0_D(a, s)$ is defined then for all extensions s' of s, $\Phi(a, s')$ is defined.
- If f is true w.r.t. (or false w.r.t.) $\Phi^0_D(a, s)$ then f is true w.r.t. (or respectively, false w.r.t.) $\Phi(a, s')$, for any extension s' of s. □

3.4 Truth of temporal formula with respect to trajectory of a-states: an approximate notion

In Section 2.2 we defined the truth of temporal formulas with respect to a trajectory of states. We now define an approximate 3-valued notion of truth with respect to a trajectory of a-states. Unlike the definition in Section 2.2 where an LTL formula is either true or false with respect to a trajectory of states, in the following true an LTL formula may evaluate to $true$, $false$ or $unknown$. Let $\sigma = \langle s_0, s_1 \ldots s_n, \ldots \rangle$ be a sequence of a-states.

1. If f is a fluent: f is said to be $true$ w.r.t (s_j, σ) if $f \in s_j$ and f is said to be $false$ w.r.t (s_j, σ) if $\neg f \in s_j$.
2. If f is an LTL formula: $\neg f$ is said to be $true$ w.r.t (s_j, σ) if f is $false$ w.r.t (s_j, σ), and $\neg f$ is said to be $false$ w.r.t (s_j, σ) if f is $true$ w.r.t (s_j, σ).
3. If f and g are LTL formulae: $f \wedge g$ is said to be $true$ w.r.t (s_j, σ) if both f and g are $true$ w.r.t (s_j, σ); and $f \wedge g$ is said to be $false$ w.r.t (s_j, σ) if either f or g is $false$ w.r.t (s_j, σ).
4. If f and g are LTL formulae: $f \vee g$ is said to be $true$ w.r.t (s_j, σ) if either f or g is $true$ w.r.t (s_j, σ); and $f \vee g$ is said to be $false$ w.r.t (s_j, σ) if both f and g are $false$ w.r.t (s_j, σ).
5. If f is an LTL formula: $\bigcirc f$ is said to be $true$ w.r.t (s_j, σ) if f is $true$ w.r.t (s_{j+1}, σ); and $\bigcirc f$ is said to be $false$ w.r.t (s_j, σ) if f is $false$ w.r.t (s_{j+1}, σ).
6. If f is an LTL formula: $\Box f$ is said to be $true$ w.r.t (s_j, σ) if f is $true$ w.r.t (s_k, σ), for every $k \geq j$; and $\Box f$ is said to be $false$ w.r.t (s_j, σ) if f is $false$ w.r.t (s_k, σ), for some $k \geq j$.

7. If f is an LTL formula: $\Diamond f$ is said to be $true$ w.r.t (s_j, σ) if f is $true$ w.r.t (s_k, σ), for some $k \geq j$; and $\Diamond f$ is said to be $false$ w.r.t (s_j, σ) if f is $false$ w.r.t (s_k, σ), for every $k \geq j$.

8. If f and g are LTL formulae: $f \cup g$ is $true$ w.r.t (s_j, σ), if there exists $m > j$ such that g is $true$ w.r.t (s_m, σ) and f is $true$ w.r.t (s_k, σ), for every k, $j \leq k < m$; and $f \cup g$ is said to be $false$ w.r.t (s_j, σ), if for every $m > j$ such that g is $true$ w.r.t (s_m, σ), there exists k, $j \leq k < m$: f is $false$ w.r.t (s_k, σ).

9. For any LTL formula if f is neither $true$ w.r.t (s_j, σ) nor $false$ w.r.t (s_j, σ), we then say that it is $unknown$ w.r.t (s_j, σ).

The conformant way to define the truth of a temporal formula w.r.t. a trajectory of a-states is to define a notion of 'extensions' of such trajectories which consist of only states (instead of a-states) and consider the truth of the given temporal formula w.r.t. all the extensions. But even though the alternative conformant definition is simpler to define, the definition that we have presented leads to a polynomial time verification of LTL formulas with respect to trajectories of a-states, and as we will show it is sound with respect to the conformant definition.

Definition 5 (Extension of trajectory) Let $\sigma = \langle s_0, s_1, \ldots \rangle$ be a trajectory of a-states. We say a trajectory of states $\sigma' = \langle s'_0, s'_1, \ldots \rangle$ is an extension of σ if for all i, s'_i is an extension of s_i. □

Proposition 2 *[Soundness]* Let $\sigma = \langle s_0, s_1, \ldots \rangle$ be a trajectory of a-states. An LTL formula f is $true$ (or $false$) with respect to (s_j, σ) implies that it is $true$ (or $false$ respectively) with respect to $(s'_j \sigma')$ for all extensions σ' of σ and the corresponding extension s'_j (of s_j from σ) in σ'. □

Proof: (sketch) We define a notion of the depth of an LTL formula. Simple fluents have depth 0, if a temporal formula f is made up of an unary operator and another formula f' then the depth of f = 1 + depth of f', and if a temporal formula f is made up of a binary operator and two formulas f_1 and f_2 then depth of f = 1 + max(depth of f_1, depth of f_2). The proof is then based on doing straightforward induction on the depth of LTL formulas. □

The above proposition shows that our approximate definition of the truths of LTL formulas with respect to trajectories of a-states is sound with respect to the alternative conformant definition based on extensions of the given trajectory of a-states. It is not complete though, as simple formulas of the form $f \vee \neg f$ will have the truth value $unknown$ in a state where f is unknown, while it will be evaluated to $true$ in each of the extensions. Having the benefit of a lower complexity of verification – which matches with the complexity of 0-approximation, at the cost of sacrificing completeness seems to us as an acceptable trade off.

3.5 Approximate planning with LTL goals and its soundness

We now define a notion of 0-approximate planning with respect to LTL goals and an incomplete initial state.

Let s be an a-state (incomplete initial state) designated as the initial state, let $a_1, \ldots, a_n$ be a sequence of deterministic actions, whose effects are described by a domain description D. The trajectory corresponding to s and $a_1, \ldots, a_n$ is the sequence of a-states $s_0, s_1, \ldots,$ that satisfies the following conditions: (i) $s = s_0$, (ii) $s_{i+1} = \Phi^0_D(a_{i+1}, s_i)$, for $0 \leq i \leq n-1$, and (iii) $s_{j+1} = s_j$, for $j \geq n$.

We say that the sequence of actions $a_1, \ldots, a_n$ is a 0-approximate plan from the initial a-state s for the LTL goal f, if f is $true$ w.r.t. (s, σ), where σ is the trajectory corresponding to s and $a_1, \ldots, a_n$.

Theorem 3.4 (Soundness of 0-approximate planning) Let D be a domain description, s_0 be an a-state, and G be an LTL goal. If a sequence of actions $a_1, \ldots, a_n$ is a 0-approximate plan from s_0 for the goal G then it is also a conformant plan from s_0 for the goal G. □

Proof
Directly follows from the soundness of Φ_D^0 with respect to Φ_D (Proposition 1) and from Proposition 2. □

3.6 Complexity of 0-approximate planning with LTL goals

Theorem 3.5 (Complexity of 0-approximate planning) Given a domain description D, an (possibly incomplete) initial state s_0 and a temporal goal G, the 0-approximate planning problem is NP-complete. □

Proof (sketch)
From [2] we already know that the 0-approximate planning problem is NP-complete for incomplete initial states and for the simplest possible case of LTL-goals: namely, for goals which are represented simply by fluent formulas. Therefore, to prove that the general problem of 0-approximate planning under incomplete initial state and LTL goals is NP-complete, it is sufficient to prove that this general problem belongs to the class NP. In other words we need to show that plan verification in this case is polynomial.

This can be shown by taking a goal statement, say for example $G \equiv \Diamond\Box f \wedge \Box\neg f$, and breaking it to intermediate goals: $G_1 := f$, $G_2 := \Box G_1$, $G_3 := \Diamond G_2$, $G_4 := \neg f$, $G_5 := \Box G_4$, $G_6 := G_3 \wedge G_5$. It is easy to see that the number of such intermediate statements is bounded by the length $|G|$. Based on the values of the fluents in each a-state in the trajectory, we can now sequentially compute the values of all these intermediate statements at all a-states. If we considering plans of length n, then we need to consider $n + 1$ a-states. When an intermediate goal is defined using propositional connectives, such as $G_6 := G_3 \wedge G_5$ computing G_6 given the value of G_3 and G_5 will take 1 logical operations in each a-state and hence $n+1$ operations total. When an intermediate goal is defined using temporal connectives ($\Box$, $\Diamond$ or U), such as $G_9 := G_8 \mathsf{U} G_7$, to compute the truth value of G_9 in any a-state, we may need in the worst case – this happens when G_9 evaluates to $false$ – $(n + 1)^2$ operations, and hence $(n + 1)^3$ operations in total. Since we are only looking for plans of polynomial length (i.e., $n \leq p(|D| + |G|)$ for some polynomial $p(x)$), we thus need a polynomial number of steps to verify a plan. □

4 Planning with Temporal knowledge goals: complexity and approximation

So far we have considered only LTL goals. When dealing with incompleteness, and specially in presence of sensing actions, it becomes natural to consider goals that also involve the knowledge aspect. Although the need for mixing temporal aspects with knowledge aspects in representing goals has been pointed out in [9, 3], we are unaware of work where such a language is formally defined. In this section we define

such a language, discuss goals whose specification requires such a rich language, and explore planning with respect to such goals.

The language we propose is a simple one where knowledge formulas are defined first and temporal operators are only applied on top of them. Thus we do not allow knowledge operators on top of temporal formulas. For example $K\Box f$ is not a valid formula in our language. Nevertheless, our language is powerful enough to express most of the goal notions in the literature that involve both knowledge and temporal aspects.

A knowledge proposition is of the form $K\varphi$, where φ is a propositional formula. A knowledge propositional formula is made up of propositions and knowledge propositions using the propositional connectives $\neg$, $\vee$, and $\wedge$. We now define temporal knowledge formulas (TKFs).

Definition 6 TKFs
(i) Knowledge propositional formulas are TKFs.
(ii) If δ and δ' are TKFs then so are $\delta \vee \delta'$, $\delta \wedge \delta'$, $\neg\delta$, $\bigcirc\delta$, $\Box\delta$ $\Diamond\delta$, and $\delta \mathsf{U} \delta'$. $\Box$

We now give some examples of goals expressed using TKFs.

1. Suppose we would like to represent the goal that the truth value of f is always known. This can be expressed as $\Box(Kf \vee K\neg f)$. A practical example of this is when f denotes 'baby is crying' and a parent has the goal of always knowing whether the baby is crying or not.

2. Another goal would be to reach a state (and stay there) where the truth value of f is known, but without changing it in the process. This can be expressed as follows:

   $\Diamond\Box(Kf \vee K\neg f) \wedge \Box(f \Rightarrow \bigcirc f) \wedge \Box(\neg f \Rightarrow \bigcirc\neg f)$

   The above corresponds to the 'sense but do not destroy' notion in [9].

3. In conjunction with the previous goal we may have the condition that a literal g is allowed to change during the execution of the plan but its final value is known and is the same as its initial value (before the plan is executed). Such requirements are part of specifying goals of diagnostic plans [3] referred to there as fixable literals. The intuition is that g may refer to properties involved in disassembling of a machine that is being diagnosed, and we would like those properties to be reverted back to their original value after the diagnosis is done. This additional condition can be expressed as:

   $(g \Rightarrow \Diamond\Box Kg) \wedge (\neg g \Rightarrow \Diamond\Box K\neg g)$

4. Similarly in conjunction with goal (2) we may have the condition that h is allowed to change during the execution of the plan but its final value is known and is the same as its initial value (before the plan is executed) or $false$. Such requirements are also part of specifying goals of diagnostic and repair plans [3], where h may refer to the abnormality of a component and hence we are allowed to fix it (make ab false) but not to break it. This additional condition can be expressed as:

   $((h \Rightarrow \Diamond\Box Kh) \wedge (\neg h \Rightarrow \Diamond\Box K\neg h)) \vee \Diamond\Box K\neg h$

The last three goals are from the literature, but in those papers they are expressed using specialized notation. They do not use a general purpose logic as we propose here.

To properly reason about the modality K it is necessary to extend the notion of states to knowledge states (or k-states) [19] where a distinction between the real state of the world, and the agent's knowledge about the world is made. A k-state is a pair $\langle s, \Sigma \rangle$ where s denotes the real state of the world, and Σ is a set of states that the agent thinks it may be in.

Given a k-state $\langle s, \Sigma \rangle$ a propositional formula f is said to be true (resp. false) in $\langle s, \Sigma \rangle$ if f holds (resp. does not hold) in s. A knowledge proposition Kf is said to be true (resp. false) in $\langle s, \Sigma \rangle$ if f holds (resp. does not hold) in all states in Σ. Given a trajectory of k-states the truth of a TKF is defined in a similar manner to Definition 2.2 using the truth of propositional formulas and knowledge formulas as defined above as the base case.

4.1 Planning with TKF goals

We now proceed towards defining planning with respect to TKFs. First, planning with respect to TKFs and in presence of incompleteness may involve special kind of actions referred to as knowledge producing or sensing actions [18]. The effects of sensing actions are expressed using propositions of the form a **determines** f, which means that after the execution of action a, the truth value of the fluent f become *known* to the agent.

The executions of actions (both sensing and otherwise) result in transition from one k-state to another. To start with the transition between k-states due to actions, which we will denote by Ψ_D, is defined using the Φ_D function defined earlier. Assuming that sensing actions do not affect the real world Ψ_D can be defined as follows [19]:

For a non-sensing action a,
$\Psi_D(a, \langle s, \Sigma \rangle) = \langle \Phi_D(a, s), \{\Phi_D(a, s') \mid s' \in \Sigma\} \rangle$.

For a sensing action a which senses the fluents $f_1, \ldots, f_k$,
$\Psi_D(a, \langle s, \Sigma \rangle) = \langle s, \{s' \in \Sigma \mid \forall i (1 \leq i \leq k)\ f_i \text{ holds in } s \text{ iff } f_i \text{ holds in } s'\} \rangle$.

From the planning perspective the next issue is to define an initial k-state corresponding to information in a given domain description D. If s is the incomplete initial state corresponding to D then an initial k-state corresponding to s would be any pair $\langle s', \Sigma \rangle$ where s' is an extension of s and Σ is the set of all extensions of s.

Now plans in presence of sensing actions [10, 19] may involve conditional statements conditioned on knowledge gained by sensing actions that precede the conditional statement. Note that the knowledge is gained during execution time and not during planning time; hence the need for conditional statements. Nevertheless, a given possible plan consisting of conditional statements when executed in a k-state goes through a trajectory of k-states. A TKF goal can then be evaluated against such trajectory of k-states. (A formal definition of this is given in [19].) We can now define the notion of a plan.

Definition 7 Let D be a domain description and G be a TKF goal. Let $\langle s_1, \{s_1, \ldots, s_l\} \rangle, \ldots \langle s_l, \{s_1, \ldots, s_l\} \rangle$ be the set of initial k-states w.r.t. D. We say a given possible conditional plan P is a plan w.r.t. D and G, if G evaluates to true w.r.t. all trajectories obtained when P is applied to the initial k-states $\langle s_1, \{s_1, \ldots, s_l\} \rangle, \ldots \langle s_l, \{s_1, \ldots, s_l\} \rangle$. □

Here while analyzing complexity of planning we consider plans whose individual execution sequences (but not necessarily the total length of the plan in terms of number of words it has) are polynomial in

the size of domain description and goal. The results related to complexity of planning are the same as they are in [2] even if we consider TKF goals. Here we present one such result. But first we need the following definition.

Definition 8 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 a possible plan is k-bounded if it has no more than k sensing actions. □

Theorem 4.1 For a given positive integer k, with incomplete information about the initial state and with k-limited sensing actions checking the existence of a k-bounded plan is Σ_2P-complete. □

Proof: Straightforward extension of the proof of Theorem 6 in [2].

4.2 0-approximate planning with TKF goals

We now briefly discuss 0-approximate planning with TKF goals. For this we first need to define the transition between a-states due to actions. While the transition remains the same for non-sensing actions (as defined by Φ_D^0), applying a sensing action a to an a-state s results in a *set of a-states* each of which can be obtained by simply adding, to the a-state, the fluent literals that may turn out to be true as a result of this sensing action.

Now when we evaluate (during planning time) a possible plan consisting of sensing actions and conditional statements against an a-state we obtain multiple trajectories of a-states. TKF goals now need to be evaluated with respect to these trajectories. The evaluation is very similar to the one in Section 3.4. Note that even now Kf is 2-valued. If f is true then Kf is true, but if f is false or unknown then Kf is false. We now define 0-approximate plan w.r.t. TKF goals, and show the soundness of this notion.

We say that a possible conditional plan is a 0-approximate plan w.r.t. D for the TKF goal f, if f is $true$ w.r.t. all pairs (s, σ), where s is the initial a-state corresponding to D, and σ is a trajectory obtained when evaluating P with respect to s.

Theorem 4.2 (Soundness of 0-approximate planning) Let D be a domain description, and G be an TKF goal. If P is a 0-approximate plan w.r.t. D for the goal G then it is also a plan from w.r.t. D for the goal G. □

Proof: (sketch) The proof is similar to the proofs of Propositions 4, 5 and 6 of [19].

As before the results related to complexity of planning are the same as they are in [2] even if we consider TKF goals. Here we present one such result.

Theorem 4.3 For a given positive integer k, with incomplete information about the initial state and with k-limited sensing actions checking the existence of a k-bounded 0-approximate plan is NP-complete. □

Proof: Straightforward extension of the proof of Theorem 7 in [2].

5 Planning with CTL and CTL* goals: complexity and approximation studies

5.1 Goal representation using branching time temporal logic

In Section 2.2 we discussed specifying planning goals using an LTL with future operators, and cited earlier work on this. In this section we consider use of a branching temporal logic in specifying planning goals that can not be specified using LTLs. The necessity of branching time operators arises when we want to specify conditions on other paths[3] starting from the states in the main path that the agent's plan suggests. For example, a robot going from position A to position B may be required to take a path so that from any point in the path there is a charging station within two steps. Note that these two steps do not have to be in the path of the robot. This goal can not be expressed using LTLs. We propose to use the branching time logic CTL* for this purpose. We now give the syntax and semantics for CTL* [6].

There are two kinds of formulas in CTL*: state formulas and path formulas. Normally state formulas are properties of states while path formulas are properties of paths. The syntax of state and path formulas is as follows. Let $\langle p \rangle$ denote an atomic proposition, $\langle sf \rangle$ denote state formulas, and $\langle pf \rangle$ denote path formulas.

$$\langle sf \rangle ::= \langle p \rangle \mid \langle sf \rangle \wedge \langle sf \rangle \mid \langle sf \rangle \vee \langle sf \rangle \mid \neg\langle sf \rangle \mid \mathsf{E}\langle pf \rangle \mid \mathsf{A}\langle pf \rangle$$
$$\langle pf \rangle ::= \langle sf \rangle \mid \langle pf \rangle \ \mathsf{U} \ \langle pf \rangle \mid \neg\langle pf \rangle \mid \langle pf \rangle \wedge \langle pf \rangle \mid \langle pf \rangle \vee \langle pf \rangle \mid \bigcirc\langle pf \rangle \mid \Diamond\langle pf \rangle \mid \Box\langle pf \rangle$$

The new symbols A and E are the branching time operators meaning 'for all paths' and 'there exists a path' respectively. As the qualification 'branching time' suggests, specification in the branching time logic CTL* are evaluated with respect to the branching structure of the time. The term 'path' in the meaning of A and E refers to a path in the branching structure of time. The branching structure is specified by a transition relation R between states of the world. Intuitively, $R(s_1, s_2)$ means that the state of the world can change from s_1 to s_2 in one step. Given a transition relation R and a state s, a path in R starting from s is a sequence of states $s_0, s_1, \ldots$ such that $s_0 = s$, and $R(s_i, s_{i+1})$ is true.

When planning in an environment where our agent is the only one that can make changes to the world, $R(s_1, s_2)$ is true if there exists an agent's action a such that $s_2 = \Phi(s_1, a)$. If there are external agents other than our agent then $R(s_1, s_2)$ is true if there exists an action (by some agent) a such that $s_2 = \Phi(s_1, a)$. We now give the formal semantics of CTL*.

Formal semantics: Semantics of CTL* formulas are defined depending on whether they are state formulas or path formulas. The truth of state formulas are defined with respect to a pair (s_j, R), where s_j is a state and R is the transition relation. In the following p denotes a propositional formula sf_is are state formulas and pf_is are path formulas.

- $(s_j, R) \models p$ if p is true in s_j.
- $(s_j, R) \models sf_1 \wedge sf_2$ if $(s_j, R) \models sf_1$ and $(s_j, R) \models sf_2$.
- $(s_j, R) \models sf_1 \vee sf_2$ if $(s_j, R) \models sf_1$ or $(s_j, R) \models sf_2$.

[3] An alternate use of branching time logic in specifying plans goals in presence of actions with non-deterministic effects has been proposed in [17]. There use of branching time logic is very different from ours. For example, in their formulation A means all possible paths that arise due to the non-determinism of actions.

- $(s_j, R) \models \neg sf$ if $(s_j, R) \not\models sf$.
- $(s_j, R) \models \mathsf{E}\, pf$ if there exists a path σ in R starting from s_j such that $(s_j, R, \sigma) \models pf$.
- $(s_j, R) \models \mathsf{A}\, pf$ if for all paths σ in R starting from s_j we have that $(s_j, R, \sigma) \models pf$.

The truth of path formulas are defined with respect to a triplet (s, R, σ) where σ given by the sequence of states $s_0, s_1, \ldots,$ is a path, R is a transition relation and s is a state in σ.

- $(s_j, R, \sigma) \models sf$ if $(s, R) \models sf$.
- $(s_j, R, \sigma) \models pf_1 \mathsf{U}\, pf_2$ iff there exists $k \geq j$ such that $(s_k, R, \sigma) \models pf_2$ and for all i, $j \leq i < k$, $(s_i, R, \sigma) \models pf_1$.
- $(s_j, R, \sigma) \models \neg pf$ iff $(s_j, R, \sigma) \not\models pf$.
- $(s_j, R, \sigma) \models pf_1 \wedge pf_2$ iff $(s_j, R, \sigma) \models pf_1$ and $(s_j, R, \sigma) \models pf_2$.
- $(s_j, R, \sigma) \models pf_1 \vee pf_2$ iff $(s_j, R, \sigma) \models pf_1$ or $(s_j, R, \sigma) \models pf_2$.
- $(s_j, R, \sigma) \models \bigcirc pf$ iff $(s_{j+1}, R, \sigma) \models pf$
- $(s_j, R, \sigma) \models \Box pf$ iff $(s_k, R, \sigma) \models pf$, for all $k \geq j$.
- $(s_j, R, \sigma) \models \Diamond pf$ iff $(s_k, R, \sigma) \models pf$, for some $k \geq j$.

We now define when a sequence of actions $a_1, \ldots, a_n$ is a plan with respect to a given initial state s and a goal in CTL*. As in the case of LTL goals in Section 2.2 we use the notion of a trajectory corresponding to s and $a_1, \ldots, a_n$.

We say a sequence of actions $a_1, \ldots, a_n$ is a plan with respect to the initial state s_0 and a goal G if $(s_0, R, \sigma) \models G$, where σ is the trajectory corresponding to s_0 and $a_1, \ldots, a_n$. (Note that a trajectory corresponding to s_0 and $a_1, \ldots, a_n$ – as defined in Section 2.2 – is a path.)

Although state formulas are also path formulas, since the evaluation of state formulas do not take into account the trajectory suggested by a prospective plan, often the overall goal of a planning problem is better expressed as a path formula which is not a state formula. Similar to an LTL goal which is just a propositional formula, a CTL* goal which is a state formula either leads to no plans (if the initial state together with R does not satisfy the goal) or leads to the plan with no actions (if the initial state together with R satisfies the goal). But unlike propositional goals in LTL, a state formula in CTL* is useful in specifying the existence of a plan.

5.2 The branching time temporal logic CTL

CTL is a branching time logic that is a subset of CTL* and has better computational properties than both LTL and CTL* for certain tasks. Unlike CTL*, CTL does not include LTL. Syntactically, a CTL formula is defined as follows:

1. Atomic propositions are CTL formulas.
2. If f_1 and f_2 are CTL formulas so are $\neg f_1$, $f_1 \wedge f_2$, $f_1 \vee f_2$, $\mathsf{A}\bigcirc f_1$, $\mathsf{E}\bigcirc f_1$, $\mathsf{A}\Box f_1$, $\mathsf{E}\Box f_1$, $\mathsf{A}\Diamond f_1$, $\mathsf{E}\Diamond f_1$, $\mathsf{A}(f_1 \mathsf{U} f_2)$, $\mathsf{E}(f_1 \mathsf{U} f_2)$.
3. Nothing else is a CTL formula.

It is easy to see that CTL formulas are state formulas and hence can only lead to empty plans or no plans and hence are not appropriate for specifying planning goals. They are useful in specifying the existence of a plan though.

We now give several examples of planning goals expressed using CTL and CTL*.

5.3 Examples of planning goals in CTL and CTL*

We start with the story of planning a route from city A to city B. Our planning goal is to find a plan to travel from A to B. We have several intermediate stopping areas between A and B and some of them have a utility center with gas, food, etc and are marked by p. We now express several goals that put conditions on paths from A to B using CTL and CTL*.

1. Suppose our goal is to get to B such that from any where in the path we can get to a state where p holds in at most two steps. This can be expressed by the following path formula (which is not a state formula) in CTL*.

   $(p \vee \mathsf{E}\bigcirc p \vee \mathsf{E}\bigcirc \mathsf{E}\bigcirc p)\ \mathsf{U}\ at_B$

   The above is not a CTL formula. Now the condition that such a path exists can be specified by the following path formula which is also a state formula.

   $\mathsf{E}\,((p \vee \mathsf{E}\bigcirc p \vee \mathsf{E}\bigcirc \mathsf{E}\bigcirc p)\ \mathsf{U}\ at_B)$

   The above is a CTL (and hence a CTL*) formula. If we use the above formula as a goal in our planning then either we get an empty plan implying that a plan exists, or get no plans when none exists. Note that in the first case we do not get the plan, but only a confirmation that a plan exists. This is because our goal is a state formula. But constructive model checkers while verifying the existence may also return a 'witness' which can lead to the plan.

2. Consider the goal of finding a path to home, such that from every point in the path there is a path to a telephone booth. This can be expressed by the following path formula (which is not a state formula) in CTL*.

   $(\mathsf{E}\Diamond\ has_telephone_booth)\ \mathsf{U}\ at_home$

   The above is not a CTL formula. But the existence of such a plan expressed as $\mathsf{E}((\mathsf{E}\Diamond\ has_telephone_booth)\ \mathsf{U}\ at_home)$ is a CTL formula.

3. Consider the goal of finding a path that travels through ports until a port is reached from where there are paths to a fort and a hill. This can be expressed by the following path formula (which is not a state formula) in CTL*.

   $is_a_port\ \mathsf{U}\ (is_a_port \wedge (\mathsf{E}\Diamond\ has_fort) \wedge (\mathsf{E}\Diamond\ has_hill))$

The above is not a CTL formula. But the existence of such a plan expressed as $\mathsf{E}(is_a_port\ \mathsf{U}\ (is_a_port \wedge (\mathsf{E}\Diamond\ has_fort) \wedge (\mathsf{E}\Diamond\ has_hill)))$ is a CTL formula.

4. Consider the goal of finding a path to a place with a hotel such that from any point in the path there is a path to a garage, until we reach a shopping center from where there is a path to the hotel. This can be expressed by the following path formula (which is not a state formula) in CTL*.

   $((\mathsf{E}\Diamond\ has_garage)\ \mathsf{U}\ (shopping_center \wedge \Diamond\ has_hotel)) \wedge \Diamond\Box\ has_hotel$

   The above is not a CTL formula.

5. Consider specifying the goal of a robot to reach a state satisfying the property h such that the states on the path are obstacle free and at least one immediate successor state has a power socket. This can be expressed by the following path formula (which is not a state formula) in CTL*.

   $(obstacle_free \wedge (\mathsf{E} \bigcirc has_power_socket))\ \mathsf{U}\ h$

   The above is not a CTL formula. But the existence of such a plan expressed as $\mathsf{E}((obstacle_free \wedge (\mathsf{E} \bigcirc has_power_socket))\ \mathsf{U}\ h)$ is a CTL formula.

6. Suppose we would like to change the last specification such that the agent has to make sure that all (instead of at least one) immediate successor state has a power socket. This can be expressed as follows:

   $(obstacle_free \wedge (\mathsf{A} \bigcirc has_power_socket))\ \mathsf{U}\ h$

   The above is not a CTL formula. But the existence of such a plan expressed as $\mathsf{E}((obstacle_free \wedge (\mathsf{A} \bigcirc has_power_socket))\ \mathsf{U}\ h)$ is a CTL formula.

7. Consider a robot that has to reach a goal state (having the property h) but on the way it has to 'maintain' a property p. Earlier in Section 2.2 we mentioned that this can be expressed in LTL as:

   $\Box\Diamond p\ \mathsf{U}\ h$

   Now suppose we are in a domain with other agents which can randomly execute an action in between the actions of our agent and we also assume that any action that can be executed by the other agents, an action with the same transition can also be executed by our agent. In this case we want to be extra careful in maintaining p and consider the other agent's actions. For that we would like to put a condition on states that are one transition away from the planned path, as these states can be reached because of the other agent's actions. The condition we want to put is that from those states our agent can correct itself (if necessary) by executing an action to reach a state where p is true. This can no longer be specified in LTL. In CTL* this can be expressed as follows:

   $\mathsf{A}(\bigcirc(\neg p \Rightarrow \mathsf{E}(\bigcirc p)))\ \mathsf{U}\ h$

   which is equivalent to

   $\mathsf{A}(\bigcirc(p \vee \mathsf{E}(\bigcirc p)))\ \mathsf{U}\ h$

   Suppose we don't need to reach a state satisfying h but we want our robot to wander around through states that satisfy the other above mentioned properties. In that case the specification will be:

   $\Box(\mathsf{A}(\bigcirc(p \vee \mathsf{E}(\bigcirc p))))$

   Note that the specification $\mathsf{A}(\bigcirc(p \vee \mathsf{E}(\bigcirc p)))$ is not right as it is a state formula and as explained earlier either has no plans or a plan with empty action. It will not result in any plans leading to wandering.

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

Theorem 5.1 For goals expressible in Branching Temporal Logics CTL and CTL*, the planning problem is **PSPACE**-hard. □

Proof of Theorem 5.1. This proof follows the same logic as proofs of **PSPACE**-hardness of other planning problems; see, e.g., [12] and [2].

To prove that the planning problem (with CTL and CTL* goals) is **PSPACE**-hard, 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:

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';

For example, for a formula $\exists x_1 \forall x_2 F$, this construction leads to the following goal: $\mathsf{EX}(\mathsf{AX}(F))$ 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.

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

By definition of the operator $\bigcirc$ ("next"), the fact that the formula $\bigcirc G'$ 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 $\bigcirc G'$ 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 $\mathsf{E}(\bigcirc G')$ 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 planning problem with respect to CTL-goals is indeed **PSPACE**-hard. Since CTL is a subset of CTL* the planning problem with respect to CTL*-goals is also **PSPACE**-hard. □

In general the planning problem with respect to CTL and CTL*-goals are not **PSPACE**-complete, as they are not in **PSPACE**. This is because, although we limit our plan lengths to be polynomial, the semantics of A and E does not assume that the paths are of polynomial length. If we restrict the meaning of A and E to be 'for all polynomial length paths' and 'there exists a polynomial length paths' then the planning problem with respect to such goals is indeed **PSPACE**-complete. In the following we argue this.

By definition, the class **PSPACE** is formed by problems which can be solved by a polynomial-*space* algorithm. Recall 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 polynomially verifiable property, and all k quantifiers run over words of polynomial 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 with the restricted meaning of A and E, the planning problem belongs to the class **PSPACE**. Indeed, all the operators of CTL* can be then described by quantifiers over words of polynomial 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 polynomial length. Thus, the existence of a plan which satisfies a given CTL*-goal can be described by a polynomial size sequence of quantifiers running over words of polynomial length. Thus, for CTL*-goals, with the restricted meaning of A and E, the planning problem does belong to **PSPACE**. Since CTL-goals are subset of CTL*-goals the same is true for CTL-goals.

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

Theorem 5.2 For goals expressible in Branching Temporal Logics CTL and CTL*, the plan checking problem is **PSPACE**-hard. □

Proof of Theorem 5.2. Similarly to the proof of Theorem 5.1 we need to prove is the desired reduction. From the proof of Theorem 5.1, 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 5.1 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. □

5.5 Complexity of the planning problem with goals expressible in a limited variant of Branching Temporal Logic

Theorems 5.1 and 5.2 mean that allowing temporal goals from CTL and CTL* can drastically increase the computational complexity of planning. These results, however, do not necessarily mean that planning

under safety and maintainability conditions is necessarily very complex. Many such conditions can be expressed in a variant of the above language, a variant for which the planning problem is much simpler than for CTL*.

The main idea behind this variant is that in many maintainability conditions, we do not need to consider all possible paths, it is sufficient to consider paths 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.

Let us first define what it means for two paths to differ in no more than k states. In other words, let us define a notion of "distance" between the two paths. A path is a particular case of a trajectory – which was defined as an infinite sequence of states $s_0, s_1, \ldots$ To make things simpler, let us therefore define the distance between arbitrary trajectories σ and σ'.

A natural way to define such a distance is as follows: First, we define an *elementary transformation* as a transformation that changes a single state. We will consider three types of elementary transformations:

- a transformation $T_{i,s}$ that replaces i-th state in the original trajectory by a state s:

$$T_{i,s}(s_0, s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots) = (s_0, s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots);$$

- a transformation $T^+_{i,s}$ that adds a state s after the i-th state in the original trajectory:

$$T^+_{i,s}(s_0, s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots) = (s_0, s_1, \ldots, s_{i-1}, s_i, s, s_{i+1}, \ldots);$$

- a transformation T^-_i that deletes the i-th state in the original trajectory:

$$T^-_i(s_0, s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots) = (s_0, s_1, \ldots, s_{i-1}, s_{i+1}, \ldots).$$

We can then define a *distance* between the trajectories σ and σ' as the smallest number of elementary transformations that transform σ into σ'. In other words, we say that σ' is *k-close* to σ if σ' can be obtained from σ by applying no more than k transformations.

It is worth mentioning that this definition is similar to the definition of a distance between the DNA sequences in bioinformatics; see, e.g., [16].

The original branching time operators E and A – 'there exists a path' and 'for all paths" – do not take into consideration how close the corresponding paths are to the original path. Instead of these operators, we can consider, for each natural number k, the restricted versions E_k and $\mathbf{A}_k$ that only consider paths which are k-close to the original path. The formal semantics of these new operators is as follows. For every path $\sigma = (s_0, \ldots, s_j, s_{j+1}, \ldots)$, and for every j, by $\sigma_{|j}$ we denote the remaining path, i.e., the path $\sigma_{|j} = (s_j, s_{j+1}, \ldots)$ Now:

- $(s_j, R, \sigma) \models \mathsf{E}_k\, pf$ if there exists a path σ' in R starting from s_j that is k-close to $\sigma_{|j}$ and for which $(s_j, R, \sigma') \models pf$.
- $(s_j, R, \sigma) \models \mathsf{A}_k\, pf$ if for all paths σ' in R starting from s_j which are k-close to $\sigma_{|j}$, we have that $(s_j, R, \sigma') \models pf$.

(Crudely speaking, the original operators E and A can be interpreted, in these terms, as the operators E_∞ and A_∞ corresponding to infinite distance $k = \infty$.)

Please note that there is also a syntactic difference between the original branching time operators and their restricted versions:

- The original operators E and A do not use the original path. Therefore, the results $\mathsf{E}\langle pf\rangle$ and $\mathsf{A}\langle pf\rangle$ of applying these operators to a path formula $\langle pf\rangle$ are state formulas.
- In contrast, the restricted versions E_k and A_k of these operators do use the original path. Therefore, the truth value of the resulting formulas $\mathsf{E}_k\langle pf\rangle$ and $\mathsf{A}_k\langle pf\rangle$ may depend on the original path σ. Hence, the expressions $\mathsf{E}_k\langle pf\rangle$ and $\mathsf{A}_k\langle pf\rangle$ are path formulas.

Let us give an example of a natural planning statement that can be expressed in terms of these operators. Suppose that we plan a road trip on an old car, and we are concerned that the car may start leaking oil (as it used to do in the past). We therefore want to plan a trip in such a way that we are always at most one step away from a repair shop. To be more precise, we want to be sure that at any state s_j along the path, if necessary, we can, instead of going to the next step s_{j+1}, first go to a place where there is a repair shop, and then continue onto s_{j+1}.

In general, such a repair would mean a 1-step delay. However, in some cases, it may be possible to do a repair without a delay. In such cases, we simply replace the original next state s_{j+1} by a new state (with repairs) and then go on to the scheduled next state s_{j+2}.

It may be also possible, in case of emergency, to get a permission to go faster; in this case, we skip s_{j+1}, go directly to the next state s_{j+2} and do repairs there.

Let us describe this planning problem in more formal terms. Let p denote the property "has a repair shop". We want the path $\sigma = (s_0, s_1, \ldots)$ to be such that for every state s_j on this path, if this state does not have a repair shop (i.e., if p is false at this state), then it should be possible to add a "detour" state s' – for which p is true – into this path, or skip the state. The resulting path will be one of the following:

- $\sigma' = (s_0, s_1, \ldots, s_j, s', s_{j+1}, \ldots)$ – the result of inserting an additional state into the original path σ;
- $\sigma' = (s_0, s_1, \ldots, s_j, s', s_{j+2}, \ldots)$ – the result of replacing the state s_{j+1} by a new state s';
- $\sigma' = (s_0, s_1, \ldots, s_j, s_{j+2}, \ldots)$ – the result of deleting the state s_{j+1} from the original path σ.

In all three cases, σ' is obtained from σ by a single elementary transformation, so σ' is 1-close to σ. In other words, our requirement means that if p is false, then in some 1-close path, p should be true in the next moment of time. Formally, this implication can be described as $\neg p \Rightarrow \mathsf{E}_1(\bigcirc p)$, or, equivalently, as $p \vee \mathsf{E}_1(\bigcirc p)$. This property must hold for all the states until we reach the goal g. So, the final formalization of the above requirement is:

$(p \vee \mathsf{E}_1(\bigcirc p))\mathsf{U}g$

For this variant, the planning problem is not as complex as for the original language CTL*. Indeed, let us denote by CTL_v^* a variant of CTL* in which, instead of the original operators E and A, we only allow operators E_k and A_k corresponding to different distances k.

Let $K > 0$ be a positive integer. We say that an expression in this language is K*-limited* if the sum of all the distances corresponding to its operators E_k and A_k does not exceed K. For example, the above expression $(p \vee \mathsf{E}_1(\bigcirc p))\mathsf{U}g$ is 1-limited.

Theorem 5.3 Let $K > 0$ be an integer. For K-limited goals expressible in Branching Temporal Logic CTL_v^*, the planning problem is **NP**-complete. □

Theorem 5.4 Let $K > 0$ be an integer. For K-limited goals expressible in Branching Temporal Logic CTL_v^*, the plan checking problem is polynomial. □

Proof of Theorems 5.3 and 5.4. Each operator E_k and A_k deals only with paths which are k-close to the original path σ. If we have a composition of such operators, e.g., $\mathbf{E}_k\mathsf{A}_l$, then we must consider paths which are k-close to the paths which are l-close to the original path σ. Due to triangle inequality, the distance between each considered path and the original path σ cannot exceed $k + l$. In other words, for such a composite statement, it is sufficient to consider paths which are $(k + l)$-close to the original path σ.

Similarly, in general, for an arbitrary formula from CTL_v^*, it is sufficient to consider only paths whose distance from the original path σ does not exceed the sum of all the distances k corresponding to different operators E_k and A_k. In other words, for a K-limited goal, it is sufficient to consider only paths which are K-close to the original path σ. We will show that there is a polynomial number of such paths and therefore, we can simply enumerate all of them.

Let us first count the number of paths which are 1-close to the original path σ, i.e., which can be obtained from σ by a single elementary transformation. Let T be a duration of the path before it reaches the final goal, and let A be the total number of possible actions. For each of T states s_i on the path, we have the following paths:

- We have at most one path $T_i^-(\sigma)$ obtained by deleting the state s_i. We say "at most one", not "one" because it is possible that the resulting trajectory $T_i^-(\sigma) = (s_0, \ldots, s_{i-1}, s_{i+1}, \ldots)$ is not a path, i.e., that no action can lead us from s_{i-1} directly to s_{i+1}.
- We have paths $T_{i,s}(\sigma) = (s_0, \ldots, s_{i-1}, s, s_{i+1}, \ldots)$ obtained by replacing the state s_i by a new state s. Each such path corresponds to a different action a applied to the state s_{i-1}. Therefore, the total number of such paths cannot exceed the total number A of possible actions.
- We also have paths $T_{i,s}^+(\sigma) = (s_0, \ldots, s_{i-1}, s_i, s, s_{i+1}, \ldots)$ obtained by inserting a new state s after the state s_i. Each such path corresponds to a different action a applied to the state s_i. Therefore, the total number of such paths cannot exceed the total number A of possible actions.

Adding up these numbers, we conclude that there are no more than $1 + A + A = 2A + 1$ paths which differ from σ in the state s_i. We have $\leq (2A + 1)$ such paths for each of T states, so the total number of 1-close paths does not exceed $T \cdot (2A + 1)$. In other words, the total number of 1-close paths is $O(T \cdot A)$.

Similarly, there exist no more than $O((T \cdot A)^2)$ paths which are 2-close to the original path, no more than $O((T \cdot A)^3)$ paths which are 3-close to the original path, etc. In general, whatever number K we fix, there is only a polynomial number $(O(T \cdot A)^K)$ of possible paths which are K-close to the original path.

Therefore, for fixed K, we can explicitly describe the new operators E_k and A_k by enumerating all such possible paths. Thus, similarly to the proof of Theorems 3.1 and 3.2, we can conclude that for planning with K-limited goals, plan checking is polynomial and the corresponding planning problem is **NP**-complete. □

6 Conclusions

In this paper we discussed the usefulness of temporal logics in expressing planning goals and precisely defined what it means for a sequence of actions or a conditional plan structure to be a plan with respect to such planning goals. We then analyzed the complexity of planning with respect to goals represented using these logics. We considered three such logics: linear temporal logic with future operators (LTL); a knowledge-temporal logic; and the branching time temporal logic CTL*.

In case of LTL goals we considered two cases: when the knowledge about the initial state is complete and when it is incomplete. In both cases we conclude that the use of linear temporal operators *does not* increase the complexity over the case when linear temporal operators are not used. Also, as in the case of planning with simple propositional (non-temporal) goals, the complexity of planning with LTL goals increases by one level when we abandon the assumption that the knowledge about initial state is complete and do conformant planning. We show how to bring down the complexity by considering an approximate notion which we show to be sound.

When planning with respect to goals expressed as knowledge-temporal formulas we conclude that the complexity is due to the knowledge aspect and not due to the temporal aspects. Thus the complexity remains the same as in the case of planning with knowledge goals and incompleteness (about the initial state).

When we allow goals which refer to *potential* future, necessitating the use of branching time temporal operators, the planning problem becomes drastically more complicated. This suggests that we should be very cautious about such more general goals. We identify a particular variant of such goals, which we refer to as K-limited goals, for which the complexity of planning reverts back to the case with simple propositional goals.

Acknowledgments

This work was supported by NASA grants NCC5-209 and NCC 2-1232, 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, CDA-9522207, ERA-0112968, and 9710940 Mexico/Conacyt.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *AAAI 96*, pages 1215–1222, 1996.

[2] C. Baral, V. Kreinovich, and R. Trejo. Planning and approximate planning in presence of incompleteness. *Artificial Intelligence Journal*, 122:241–267, 2000.

[3] C. Baral, S. McIlraith, and T. Son. Formulating diagnostic problem solving using an action language with narratives and sensing. In *KR 2000*, pages 311–322, 2000.

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

[5] G. De Giacomo and M. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Proc. of ECP 1999*, pages 226–238, 1999.

[6] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of theoretical computer science: volume B*, pages 995–1072. MIT Press, 1990.

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

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

[9] K. Golden and D. Weld. Representing sensing actions: the middle ground revisited. In *KR 96*, pages 174–185, 1996.

[10] H. Levesque. What is planning in the presence of sensing? In *AAAI 96*, pages 1139–1146, 1996.

[11] P. 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.

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

[13] R. Niyogi and S. Sarkar. Logical specification of goals. In *Proc. of 3rd international conference on Information Technology*, pages 77–82, 2000.

[14] O. Ozveren, A. Willsky, and P. Antsaklis. Stability and stabilizability of discrete event dynamic systems. *JACM*, 38(3):730–752, July 1991.

[15] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[16] P. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, 2000.

[17] M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In *IJCAI'01*, 2001.

[18] R. Scherl and H. Levesque. The frame problem and knowledge producing actions. In *AAAI 93*, pages 689–695, 1993.

[19] T. Son and C. Baral. Formalizing sensing actions: a transition function based approach. *Artificial Intelligence*, 125(1-2):19–93, 2001.

[20] D. Weld and O. Etzioni. The first law of robotics (a call to arms). In *AAAI*, pages 1042–1047, 1994.