Reasoning about Actions and Planning in LTL Action Theories

Diego Calvanese, Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it

Moshe Y. Vardi
Department of Computer Science
Rice University, P.O. Box 1892
Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu

Abstract

In this paper, we study reasoning about actions and planning with incomplete information in a setting where the dynamic system is specified by adopting Linear Temporal Logic (LTL). Specifically, we study: (i) reasoning about action effects (i.e., projection, historical queries, etc.), in such a setting; (ii) when actions can be legally executed, assuming a non-prescriptive approach, where executing an action is possible in a given situation unless forbidden by the system specification; (iii) the problem of finding conformant plans for temporally extended goals that consist of arbitrary LTL formulas, thus allowing for expressing sophisticated dynamic requirements. For each of these problems we establish techniques and characterize the computational complexity. For the last two problems we make use of a second-order variant of LTL.

1 INTRODUCTION

Linear Temporal Logic (LTL) is a linear-time temporal logic which has been widely used for specifying and verifying properties of dynamic systems, such as safety, liveness, fairness, etc. [26, 13, 39]. In this paper, we study reasoning about actions and planning in a setting where we have incomplete information on the dynamic system and our knowledge on it is represented in LTL. This means that we represent the behavior of the system as a set of sequences of situations, where transitions from one situation to the next are caused by actions. In particular, we describe the system by introducing a set of atomic facts, called fluents, whose truth value changes as the system evolves, and by specifying through a logic (LTL, in our case) the effects of actions on such a set of facts. Such an approach is shared by several proposals for reasoning about actions, e.g., [4, 24, 12]. Here, however, we do not focus on specific action theories for the dynamic system. Instead, we allow the system specification to be an arbitrary LTL formula.

First, we address reasoning about action effects in the above setting. In particular, we focus on problems of the following form: given a finite sequence of actions, determine whether a certain property holds (which include projections and historical queries [28]).

Next, we address executability of actions in a linear-time setting. We observe that linear-time logics cannot be used for prescriptively asserting which actions are possible in a given situation. This is different from branching-time formalisms, such as the situation calculus [28], the fluent calculus [34], dynamic logics [10], and branching-time temporal logics [7, 9], where executability can be expressed directly in the language. There are some workarounds to this problem in the linear-time setting, but they involve action theories of a specific form, see [30] for a discussion. Here, instead, we follow a non-prescriptive approach (similar in the spirit to [16, 22, 3]): executing an action is possible in a given situation, unless forbidden by the system specification. We formalize and study such a notion in the context of LTL.

Finally, we address the problem of finding *conformant plans* for *temporally extended goals* in dynamic systems in the presence of incomplete information on the initial situation and on the full effects of actions. Conformant plans are sequences of actions that are guaranteed to fulfill the goal specification, even when we have incomplete information on the dynamic system in which the plan is to be executed [14, 17, 33, 28, 5]. Temporally extended goals are goals that specify acceptable sequences of states [1]. They subsume the usual goals expressing "reachability" of desired conditions, as well as generalized goals, such as "don't-disturb" and "restore" requirements [41]. More generally, they allow for expressing complex temporal properties typically used in the specification of processes [13, 39]. Planning for temporally extended goals has been studied in [1, 6, 2, 21], where complete information is assumed, and in [11], where the system is specified as a deterministic transition system, except for the initial situation, on which incomplete knowledge is assumed. Related kinds of goals, expressed in CTL, have been investigated in [25]. We observe that CTL [13] is a branching temporal logic whose expressive power is incomparable to that of LTL. Here we address conformant planning for temporally extended goals specified by means of arbitrary LTL formulas.

Observe that, as we deal with goals expressing general temporal properties, even sequential plans may in fact involve loops, since goals of this form may require *infinite executions*. Consider, for example, a plan to satisfy the following requirement: whenever certain triggering conditions are met within a finite (but undetermined) number of steps, a specified state of affairs must be brought about in which the triggering conditions are met again.

For each of the above three problems, i.e., reasoning about action effects, reasoning about executability of actions, and conformant planning, we establish techniques and characterize the computational complexity. To formally capture the last two problems and to derive reasoning procedures, we make use of a second-order variant of LTL.

In this paper we focus on LTL as the linear-time formalism. However, all the results presented here can be immediately rephrased in terms of more expressive linear-time formalisms, such as μLTL, which is able to express any ω-regular property [37].

2 PRELIMINARIES

We introduce the temporal logic LTL and discuss its relationship to Büchi automata on infinite strings. We also introduce an extension of LTL with second-order quantification.

2.1 LINEAR TEMPORAL LOGIC (LTL)

Linear Temporal Logic (LTL) was originally proposed in Computer Science as a specification language for concurrent programs [26]. *Formulas* of LTL are built from a set $\mathcal{P}$ of propositional symbols and are closed under the boolean operators, the unary temporal operators $\bigcirc$, $\Diamond$, and $\Box$, and the binary temporal operator $\mathcal{U}$. Intuitively, $\bigcirc\varphi$ says that φ holds at the *next* instant, $\Diamond\varphi$ says that φ will *eventually* hold at some future instant, $\Box\varphi$ says that from the current instant on φ will *always* hold, and $\varphi\,\mathcal{U}\,\psi$ says that at some future instant ψ will hold and *until* that point φ holds. In fact, it is sufficient to consider as temporal operators only $\bigcirc$ and $\mathcal{U}$, since $\Diamond\varphi$ can be viewed as an abbreviation for $\mathbf{true}\,\mathcal{U}\,\varphi$, and $\Box\varphi$ as an abbreviation for $\neg\Diamond\neg\varphi$. We additionally use the standard boolean abbreviations $\vee$ (or) and $\rightarrow$ (implies).

The semantics of LTL is given in terms of interpretations over a *linear structure*. Without loss of generality and for simplicity of presentation, we use the natural numbers $\mathbb{N}$ as the linear structure. Hence, for an instant $i \in \mathbb{N}$, the successive instant is $i+1$. An *interpretation* is a function $\pi : \mathbb{N} \rightarrow 2^{\mathcal{P}}$, which assigns to each element of $\mathcal{P}$ a truth value at each instant $i \in \mathbb{N}$. For an interpretation π, we inductively define when an LTL formula φ *is true* at an instant $i \in \mathbb{N}$, in symbols $\pi, i \models \varphi$, as follows:

- $\pi, i \models p$, for $p \in \mathcal{P}$ iff $p \in \pi(i)$.
- $\pi, i \models \neg\varphi$ iff not $\pi, i \models \varphi$.
- $\pi, i \models \varphi \wedge \varphi'$ iff $\pi, i \models \varphi$ and $\pi, i \models \varphi'$.
- $\pi, i \models \bigcirc\varphi$ iff $\pi, i{+}1 \models \varphi$.
- $\pi, i \models \varphi\,\mathcal{U}\,\varphi'$ iff for some $j \geq i$, we have that $\pi, j \models \varphi'$ and for all k, $i \leq k < j$, we have that $\pi, k \models \varphi$.

A formula φ *is true* in π, in notation $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A formula φ is *satisfiable* if it is true in some interpretation, and is *valid*, if it is true in every interpretation.

Theorem 1 [31] *Satisfiability (and validity) for* LTL *formulas are PSPACE-complete.*

2.2 BÜCHI AUTOMATA AND CORRESPONDENCE WITH LTL

There is a tight relation between LTL and Büchi automata on infinite words (see e.g., [39]).

Given a finite nonempty alphabet Σ, an *infinite word* is an element of Σ^ω, i.e., an infinite sequence $a_0 a_1 \cdots$ of symbols from Σ. A *Büchi automaton* [36] is a tuple $A = (\Sigma, S, S_0, \rho, F)$ where:

- Σ is the alphabet of the automaton.
- S is the finite set of states.
- $S_0 \subseteq S$ is the set of initial states.
- $\rho : S \times \Sigma \rightarrow 2^S$ is the transition function of the automaton (the automaton does not need to be deterministic).

- $F \subseteq S$ is the set of accepting states.

The *input words* of A are infinite words $a_0 a_1 \cdots \in \Sigma^\omega$. A *run* of A on an infinite word $a_0 a_1 \cdots$ is an infinite sequences of states $s_0 s_1 \cdots \in S^\omega$ such that $s_0 \in \mathcal{S}_0$ and $s_{i+1} \in \rho(s_i, a_i)$. A run r is *accepting* iff $lim(r) \cap F \neq \emptyset$, where $lim(r)$ is the set of states that occur in r infinitely often. In other words, a run is accepting if it gets into F infinitely many times, which in turn means, being F finite, that there is at least one state $s_f \in F$ that is visited infinitely often. The *language* accepted by A, denoted by $L(A)$, is the set of words for which there is an accepting run. Nondeterministic Büchi automata are closed under intersection and complement [36]. More precisely, for two Büchi automata A_1 and A_2, the a number of states of the automaton accepting $L(A_1) \cap L(A_2)$ is polynomial in the number of states of A_1 and A_2 [40], while the number of states of the automaton accepting $\Sigma^\omega \setminus L(A_1)$ is exponential in the number of states of A_1.

The *nonemptiness* problem for an automaton is to decide, given an automaton A, whether $L(A) \neq \emptyset$, i.e., whether the automaton accepts at least one word.

Theorem 2 [40] *Nonemptiness of Büchi automata is NLOGSPACE-complete.*

The nonemptiness algorithm in [40] actually returns a witness for nonemptiness, which is a finite prefix followed by a cycle.

Both automata on infinite words and linear time logics, such as LTL, are widely used in verification to specify properties of dynamic systems [20, 40, 38]. The two formalisms can be put in a tight correspondence by considering as alphabet of the automaton the set $2^{\mathcal{P}}$ of propositional interpretations of the propositional variables in $\mathcal{P}$. Hence, an infinite word over the alphabet $2^{\mathcal{P}}$ accepted by an automaton can be viewed as an interpretation of an LTL formula over $\mathcal{P}$.

Theorem 3 [40] *For every* LTL *formula φ one can effectively construct a Büchi automaton A_φ whose number of states is at most exponential in the length of φ and such that $L(A_\varphi)$ is the set of models of φ.*

2.3 QUANTIFIED LINEAR TEMPORAL LOGIC (QLTL)

We make also use of an extension of LTL by second-order quantifiers over propositions, called Quantified Linear Temporal Logic (QLTL) [32]. Formally, formulas of QLTL are built using the operators of LTL plus an operator for existential quantification over propositions, i.e., $\exists p.\varphi(p)$, where p is a proposition variable and $\varphi(p)$ is a QLTL formula in which p occurs free. We use also $\forall p.\varphi(p)$ as an abbreviation for $\neg\exists p.\neg\varphi(p)$. The semantics of such an operator is defined as follows

- $\pi, i \models \exists p.\varphi$ iff there is some π' that agrees with π except for the interpretation of proposition p, and such that $\pi', i \models \varphi$.

Observe that the two interpretations π and π' may disagree on the interpretation of p at any time point, not only the initial one.

Every QLTL formula can be put in *prefix normal form*, i.e., written in the form

$$Q_1 p_1.Q_2 p_2.\cdots Q_h p_h.\varphi$$

where each Q_i is either $\forall$ or $\exists$, and φ is a (quantifier-free) LTL formula. If Q_1 is $\exists$ and there are $k-1$ alternations of quantifiers, we say that the formula is a Σ_k^{QLTL} formula. If Q_1 is $\forall$ and there are $k-1$ alternations of quantifiers, we say that the formula is a Π_k^{QLTL} formula.

The following complexity characterization of satisfiability for QLTL formulas was given in [32] (for a positive integer k, k-EXPSPACE denotes the set of languages accepted by a Turing machine with space bounded by $2^{2^{\cdots 2^n}}$, where the height of the tower is k, and n is the size of the input tape).

Theorem 4 [32] *Satisfiability for Σ_{k+1}^{QLTL} and for Π_k^{QLTL} formulas, with $k \geq 1$, is k-EXPSPACE-complete.*

The result is obtained by reducing satisfiability of a QLTL formula φ to non-emptiness of Büchi automata. Intuitively, one first builds the Büchi automaton for the LTL matrix of φ. Then, existential quantification is handled by projecting out the existentially quantified propositions from the automaton, while universal quantification requires a complementation [19], which gives rise to an exponential blow-up in the number of states of the automaton.[1]

3 REASONING ABOUT ACTIONS USING LTL

We may characterize the behavior of a dynamic system by a set of evolutions, each of which can be represented as a sequence of *situations* [28]. Transitions from one situation to the next are caused by actions. If

[1] [15] describes a symbolic implementation of the complementation construction described in [19].

we have complete information on the current situation and complete information on the effects of actions, we are able to determine the actual evolution of the system corresponding to a given sequence of actions. Typically, however, we have incomplete information both on the current situation and on the actual effects of actions. So, given a sequence of actions, we will only be able to isolate a set of possible evolutions, one of which is the actual one. It follows that, if we want to check whether a certain dynamic property holds, we need to check it for every possible evolution.

Following a methodology typical of the literature on reasoning about actions in AI (cf. [28, 30, 29, 16, 35]) we specify a dynamic system by introducing a set of atomic facts, here called *fluents*, whose truth value changes as the system evolves, and by specifying, through a logical formalism, the *effects* of *actions* on such a set of facts. Then, given a dynamic property, we can use logical inference to check whether the property holds.

3.1 STRUCTURAL REQUIREMENTS

Specifically, we adopt as logical formalism LTL. Since LTL does not provide us a direct notion of action, we use propositions to denote them. Hence we consider two separate sets of atomic propositions:

- *fluents* $\mathcal{F}$, which are propositions that denote atomic facts on the current situation;
- *actions* $\mathcal{A}$, which are special propositions denoting that an action has just been performed.

We describe the dynamic system by means of a conjunction of a finite set of LTL formulas. To suitably model actions we always add as conjuncts to the specification of the dynamic system the following LTL formulas:

$$\Box\left(\bigvee_{a\in\mathcal{A}} a\right)$$

to specify that one action must be performed in order to get to a new situation, and

$$\Box\left(\bigwedge_{a\in\mathcal{A}} (a \rightarrow \bigwedge_{b\in\mathcal{A}, b\neq a} \neg b)\right)$$

to specify that a single action at a time can be performed.

In this way, a unique proposition $a \in \mathcal{A}$ holds in each situation, and it specifies the action a that has been just performed to get to that situation[2]. We include in $\mathcal{A}$ a proposition a_d, representing a dummy action with no effects. In the initial situation, where no actual action has been performed yet, we require for uniformity that a_d holds.

3.2 SPECIFYING EFFECTS

Apart from such structural requirements on modeling actions, we allow for (finite) sets of arbitrary LTL formulas in specifying the dynamic system. A simple way to specify the dynamic system is as follows:

- We describe the *initial situation* as an arbitrary propositional formula φ_{init} involving only fluents.
- We specify *effects of actions* by means of formulas of the form

  $$\Box(\varphi \rightarrow \bigcirc(a \rightarrow \psi))$$

  where ψ and φ are arbitrary propositional formulas involving only fluents. Such a formula says that executing the action a under the conditions denoted by φ brings about the conditions denoted by ψ.[3]
- We may also specify *state constraints* by means of formulas of the form

  $$\Box\phi$$

  where ϕ is a propositional formula involving only fluents.

Observe that with this formalization we may have incomplete information on the initial situation. Moreover we may have incomplete information on the effects of an action, i.e., even if we know the truth-values of all the fluents in a given situation we may not know their value after the execution of an action.

If we consider the above formalization to be too liberal and we are willing to completely specify the effects of actions, then we can use, for example, LTL formulas that correspond to Reiter's *successor state axioms* [28] (which also provide a solution to the frame problem). Namely

$$\Box(\bigcirc F \equiv \bigvee_a (\varphi_a^+ \wedge \bigcirc a) \vee (F \wedge \bigwedge_b (\neg\varphi_b^- \vee \bigcirc\neg b)))$$

where F is a fluent, the a's are those actions that under the circumstances described by the propositional formulas φ_a^+, involving only fluents, make F become true,

[2] In fact, the approach can be easily extended to deal with concurrent atomic actions.

[3] Note that the formula $\Box(\varphi \rightarrow \bigcirc(a \rightarrow \varphi))$ corresponds to a frame axiom expressing that φ does not change performing a.

and the b's are those actions that under the circumstances described by the propositional formulas φ_b^-, involving only fluents, make F become false. Hence the formula above expresses that F is true next if and only if either one of the a's is executed and φ_a^+ is currently true, or F is currently true, and none of the b's, such that φ_b^- is currently true, is executed.

If instead we think that the above formalization is too restricted, then LTL allows us also to express very loose effect specifications, such as "continuing to chop a tree sooner or later makes it fall down": $(\Diamond\Box\neg chop) \vee \Diamond falls\_down$, i.e., there cannot be a sequence of situations in which the tree is chopped infinitely often but it does not fall down.

Finally, LTL allows for expressing in a natural way several forms of *narratives* [29, 30, 27]. Indeed, an LTL formula may express that certain actions occur, or certain conditions are brought about, according to specified temporal patterns. For example, "after three steps Mary arrives to the airport, then, eventually, she boards the plane", can be expressed as $\bigcirc\bigcirc\bigcirc(arrives\_airport \wedge \Diamond boards\_plane)$.

3.3 REASONING ABOUT ACTIONS EFFECTS IN LTL

The discussion above shows that LTL is well suited to describe dynamic systems wrt action effects. In this paper, however, we do not focus on specific ways to formalize the dynamic system. Instead, as mentioned, we allow for any LTL formula as a description of the dynamic system, as long as the structural requirements are enforced.

We can use LTL validity to reason about action effects, i.e., to solve problems of the following form: given a finite sequence of actions, determine whether a certain property holds [28]. Let Γ be the formula describing the dynamic system, and let us introduce the formula

$$\begin{aligned} &Occurs(a_0 \cdots a_k, rs) \doteq \\ &\quad (a_0 \wedge \bigcirc(a_1 \wedge \bigcirc(\cdots \bigcirc(a_k \wedge rs)\cdots))) \wedge \\ &\quad \Box(rs \rightarrow \bigcirc\Box\neg rs) \end{aligned}$$

which expresses that the sequence of actions $a_0 \cdots a_k$ occurs, resulting in a situation denoted by the new proposition rs (first conjunct), and that rs is true only once (second conjunct). Note that rs acts as a marker for the situation resulting by executing $a_0 \cdots a_k$. The *projection problem* (cf. [28]), "does the property φ hold after the execution of the sequence of actions $a_0 \cdots a_k$?", can be solved by checking the validity of

$$\Gamma \rightarrow (Occurs(a_0 \cdots a_k, rs) \rightarrow \Box(rs \rightarrow \varphi))$$

Also, *Historical queries* (cf. [28]) of the form "does the property φ always hold over the duration of the sequence of actions $a_0 \cdots a_k$?" can be answered by checking the validity of

$$\Gamma \rightarrow (Occurs(a_0 \cdots a_k, rs) \rightarrow \Box((\Diamond rs) \rightarrow \phi))$$

Similarly, historical queries of the form "does the property φ hold at some point over the duration of the sequence of actions $a_0 \cdots a_k$?" can be answered by checking the validity of

$$\Gamma \rightarrow (Occurs(a_0 \cdots a_k, rs) \rightarrow \Diamond(\varphi \wedge \Diamond rs))$$

Since validity in LTL is PSPACE-complete, we have that reasoning about action effects of a finite sequence of actions is PSPACE-complete as well. More formally we can state the following theorem.

Theorem 5 *Given an* LTL *formula* Γ *specifying a dynamic system, and an* LTL *formula* Φ *specifying a dynamic property, deciding whether* $\Gamma \rightarrow \Phi$ *is PSPACE-complete.*

4 LEGAL ACTION SEQUENCES

A fundamental question that arises is what actions are actually allowed at each given point. Often, in formalisms for reasoning about actions, one adopts a *prescriptive* approach, by specifying (explicitly or implicitly[4]) the circumstances under which an action is allowed at a given situation [28].

In LTL, on the other hand, we cannot express directly in the language that a certain action *is possible.* Hence we are forced to adopt a non-prescriptive approach, i.e, it is always possible to execute an action, unless it contradicts the system specification. However, one should be careful in adopting this notion since LTL does not adequately capture causality [22] in the presence of incomplete information.

Consider for example the formula $\Box((\bigcirc a) \rightarrow F)$, which says that, if action a is performed next, then the fluent F must be true now. While this is a logically meaningful sentence, it is problematic in defining allowable actions[5]. In principle, the actual state of the world corresponds to a certain truth assignment to the fluents. An action is *executable* if performing it does not

[4]Possibly involving a solution to the *qualification problem* [23, 34].

[5]Observe that, if we try to interpret the above formula causally, we get a quite counterintuitive interpretation: performing a has the effect of making F true in the situation preceding the execution of a, i.e., a would have an effect on the past.

contradict such a truth assignment. If in the actual state of the world F is true, then we can actually execute a, while if F is false, we cannot. Now, in general, we have only partial information on the current state of the world. So we need to ensure executability of actions whatever the current state of the world actually is. In our example, if we do not know whether F is true in the current situation, then we should not ask to execute a, since this may not be possible.

This difficulty is shared by virtually all linear-time formalisms, including those developed for reasoning about actions, such as the Event Calculus. The typical way to overcome this problem is to constrain the system specification so that the *principle of directionality* is fulfilled: "information about a given time is deductively independent from information about a later time", see [30] for a detailed discussion. This requires to choose special forms of logical theories for describing the system.

Here instead, we study the case where the system specification Γ itself is not constrained in any way, and introduce a notion of *legality* of a sequence of actions wrt Γ, to characterize the sequences of actions that are allowed. We say that a sequence $a_0 \cdots a_m$ of actions and a sequence $\sigma_0 \cdots \sigma_n$ of truth assignments to the fluents in $\mathcal{F}$ are *consistent with* Γ if there exists a model of Γ whose interpretation of the actions in the first m instants coincides with $a_0 \cdots a_m$ and whose interpretation of the fluents in the first n instants coincides with $\sigma_0 \cdots \sigma_n$. An infinite sequence of actions $a_0 a_1 \cdots$ is *consistent with* Γ if there exists a model of Γ whose interpretation of the actions coincides with $a_0 a_1 \cdots$. We say that an action a_{k+1} is *legal after the sequence of actions* $a_0 \cdots a_k$ if for all sequences $\sigma_0 \cdots \sigma_k$ of truth assignments to the fluents in $\mathcal{F}$, if $a_0 \cdots a_k$ and $\sigma_0 \cdots \sigma_k$ are consistent with Γ, then also $a_0 \cdots a_k a_{k+1}$ and $\sigma_0 \cdots \sigma_k$ are consistent with Γ. The sequence a_0 is trivially legal, as long as Γ is satisfiable[6]. A finite sequence of actions $a_0 \cdots a_k$ is *legal* if a_i is legal after $a_0 \cdots a_{i-1}$, for $i = 1 \dots k$. An infinite sequence of actions $a_0 a_1 \cdots$ is *legal* if it is consistent with Γ, and if for all $k > 0$, a_k is legal after $a_0 \cdots a_{k-1}$.

The notion of legality cannot be expressed in an obvious way in LTL. However, in order to check the legality of a given finite sequence of actions wrt a system specification Γ, we can apply directly the definition of legality for finite sequences introduced above. This allows us to reduce the problem to a finite (although exponential) number of LTL satisfiability checks. It is easy to see that this provides us a PSPACE upper bound in the size of Γ and the length of the sequence.

[6] Recall that in our formalization a_0 must be the dummy action a_d.

Next we turn to infinite sequences, and we show that legality can be captured in QLTL. Let us introduce the formulas

$$\begin{aligned} \mathit{Point}(\mathit{now}) &\doteq \Diamond \mathit{now} \wedge \Box(\mathit{now} \rightarrow \bigcirc\Box\neg \mathit{now}) \\ \mathit{EqUntil}(\vec{x}, \vec{y}, \mathit{now}) &\doteq \Box((\Diamond \mathit{now}) \rightarrow \vec{x} \equiv \vec{y}) \\ \mathit{EqNext}(\vec{x}, \vec{y}, \mathit{now}) &\doteq \Box(\mathit{now} \rightarrow \bigcirc(\vec{x} \equiv \vec{y})) \end{aligned}$$

where now is a proposition that acts as a marker, $\vec{x}$ and $\vec{y}$ are tuples of variables, one for each action (fluent), and $\vec{x} \equiv \vec{y}$ stands for the conjunction of equivalences among corresponding components of the two tuples. The formula $\mathit{Point}(\mathit{now})$ expresses that now holds at a single time point. $\mathit{EqUntil}(\vec{x}, \vec{y}, \mathit{now})$ expresses that $\vec{x}$ and $\vec{y}$ coincide at every time point until the one where now holds. $\mathit{EqNext}(\vec{x}, \vec{y}, \mathit{now})$ expresses that $\vec{x}$ and $\vec{y}$ coincide at the time point following the one where now holds.

Let us use $\Gamma(\vec{a}, \vec{f})$ to denote the system specification Γ in which we have explicited all the actions $\vec{a}$ and all the fluents $\vec{f}$ as parameters.

Then we can capture the notion of legality after a sequence of actions by means of the QLTL formula

$$\begin{aligned} &\mathit{LegalNext}(\Gamma, \vec{a}, \mathit{now}) \doteq \\ &\quad \mathit{Point}(\mathit{now}) \wedge \\ &\quad \forall \vec{a}_1. \forall \vec{f}_1. \Gamma(\vec{a}_1, \vec{f}_1) \wedge \mathit{EqUntil}(\vec{a}_1, \vec{a}, \mathit{now}) \\ &\qquad\qquad \rightarrow \exists \vec{a}_2. \exists \vec{f}_2. \Gamma(\vec{a}_2, \vec{f}_2) \wedge \\ &\qquad\qquad\qquad \mathit{EqUntil}(\vec{a}_2, \vec{a}_1, \mathit{now}) \wedge \\ &\qquad\qquad\qquad \mathit{EqUntil}(\vec{f}_2, \vec{f}_1, \mathit{now}) \wedge \\ &\qquad\qquad\qquad \mathit{EqNext}(\vec{a}_2, \vec{a}, \mathit{now}) \end{aligned}$$

Intuitively, such a formula expresses that for every interpretation of the actions and fluents satisfying Γ that agrees with $\vec{a}$ till now (but possibly differs in the fluents), there exists a further interpretation satisfying Γ that agrees both in the actions and the fluents with the first one till now, in which the action performed next is the one selected by $\vec{a}$.

Finally, we can characterize legal infinite sequences of actions by means of the QLTL formula:

$$\mathit{Legal}(\Gamma, \vec{a}) \doteq \exists \vec{f}. \Gamma(\vec{a}, \vec{f}) \wedge \forall \mathit{now}. \mathit{LegalNext}(\Gamma, \vec{a}, \mathit{now})$$

Such a formula expresses that the sequence of actions resulting from the interpretation of $\vec{a}$ is consistent with Γ, and that every prefix of such a sequence continues next with a legal action.

Theorem 6 *An infinite sequence of actions $a_0 a_1 \cdots$ is legal wrt an* LTL *system specification Γ iff there exists an interpretation π interpreting the actions $\vec{a}$ according to $a_0 a_1 \cdots$ and such that $\pi \models \mathit{Legal}(\Gamma, \vec{a})$.*

Observe that, in defining $Legal(\Gamma, \vec{a})$, one alternation of second-order quantifiers is required. This is an indication that reasoning on legality is generally quite hard. Indeed, if we put $Legal(\Gamma, \vec{a})$ in prefix normal form, we get a Π_2^{QLTL} formula. Hence, considering Theorem 4 we get:

Theorem 7 *Checking the existence of an infinite sequence of actions that is legal wrt an* LTL *system specification* Γ *can be done in 2-EXPSPACE.*

Theorem 4 gives us in fact a constructive method to check the existence of a legal infinite sequence of actions using automata theoretic techniques. In particular, we start from $Legal(\Gamma, \vec{a})$ in prefix normal form. We construct a Büchi automaton A_1 corresponding to the matrix of such a formula. The automaton A_1 is exponential in the size of the matrix, and hence of Γ. Next we project out the existentially quantified variables from A_1, getting an automaton A_2. To deal with universal quantification, we complement A_2, obtaining an automaton A_3 of double exponential size, project out the universal quantified variables from A_3, and then complement again, obtaining an automaton A_4. The automaton A_4 would be of triple exponential size in Γ. However, we can do the last complementation on the fly while checking for non-emptiness, thus obtaining the 2-EXPSPACE upper bound.

It turns out that the bound in the previous theorem is actually tight.

Theorem 8 *Checking the existence of an infinite sequence of actions that is legal wrt an* LTL *system specification* Γ *is 2-EXPSPACE-hard.*

Proof sketch: Recall that satisfiability of Π_2^{QLTL} formulas is 2-EXPSPACE-complete. The lower bound proof in [32] is a reduction from Turing machines that require doubly exponential space. It cannot, however, be easily adapted to our setting. The difficulty is that in our problem Γ, which can be used to encode the Turing machine, appears on both the left-hand side and the right-hand side of an implication. The main difference between the two sides is the occurrence of the additional action described by *EqNext*. The key idea of our lower-bound proof is the introduction of a special action $a_{illegal}$ that, on one hand, cannot occur in a legal sequence of actions, while, on the other hand, guarantees that a finite sequence of actions in which $a_{illegal}$ occurs is consistent with Γ. As an example, suppose that Γ is the formula $\Diamond a_{illegal} \rightarrow p_0$. A finite sequence of actions in which $a_{illegal}$ occurs is consistent with Γ, since we can take p_0 to be true at time 0. On the other hand, the action $a_{illegal}$ cannot be legal, since it implies backward directionality.

To encode a Turing machine that requires doubly exponential space, we intend the sequence of actions to represent an accepting run of such a machine. Such a run consists of a sequence of configurations, each one of length 2^{2^n} (Γ has to be of length polynomial in n). To say that such a sequence of configurations is a proper computation of the machine, we need to "point" to cells that are distance 2^{2^n} apart and say that they are properly related. Such "pointing" is accomplished in [32] via second-order universal quantification.

Here we have implicit universal quantification over the sequence $\sigma_0, \ldots, \sigma_k$ of truth assignment to fluents in the definition of legality. In particular we have special fluents p_{first} and p_{last} that can hold at at most one point (this is ensured by Γ). The intention is for p_{last} to hold at point k. This is accomplished by requiring in Γ that $\Box(p_{last} \rightarrow \bigcirc\bigcirc a_{illegal})$. We know that in a legal sequence a_i cannot be $a_{illegal}$, for $1 \leq i \leq k+1$, so if p_{last} holds it must be at point k. We can now distinguish between the left-hand side and the right-hand side in the definition of legality. In the left-hand side, a_{k+1} is not yet defined, so it can be $a_{illegal}$. On the right-hand side a_{k+1} is an action in a legal sequence, so it cannot be $a_{illegal}$. Now we can use the left-hand side to say that p_{first} and p_{last} holds at points that are 2^{2^n} apart, and we use the right-hand side to require that the actions in these points are properly related. Further details are left to the full paper. ■

Hence, verifying existence of a sequence of actions that is legal wrt a system specification is 2-EXPSPACE-complete. To the best of our knowledge this is the hardest natural problem known for LTL.

Although not optimal from the point of view of computational complexity, we can also characterize legality of a finite sequence $a_0 \cdots a_k$ of actions, in terms of satisfiability of the QLTL formula

$$\begin{aligned} &Occurs(a_0 \cdots a_k, rs) \ \wedge \\ &\forall now.\Box(now \wedge \Diamond\bigcirc rs) \rightarrow LegalNext(\Gamma, \vec{a}, now) \end{aligned}$$

5 PLANNING

The logic LTL can express very sophisticated dynamic properties, which can be either verified wrt the system specification, but can also be used as temporally extended goals [1] to synthesize plans. For example, *reachability of a desired state of affairs*, i.e., "is a situation where a given goal φ_{goal} holds reachable?", can be expressed by the formula $\Diamond\varphi_{goal}$. Goals can also be more sophisticated. For example, an *achieving and maintenance goal*, i.e., "is there a sequence of actions that achieves a certain goal φ_{agoal} while another goal φ_{mgoal} is kept satisfied?", can be expressed by the formula $\varphi_{mgoal}\,\mathcal{U}\,\varphi_{agoal}$. Similarly, *safety*, *invariance*, *liveness*, and *fairness properties* can be expressed in LTL.

We provide a method for *conformant planning* [33] in the setting where both the dynamic system and the goal are specified by arbitrary formulas of LTL. The problem of conformant planning consists in constructing a plan, i.e., a sequence of actions, that guarantees satisfaction of the goal whenever the conditions specified for the system are satisfied. In general, a plan that satisfies an LTL formula needs to be infinite, although it is finitely representable. If the goal can be fulfilled in a finite number of steps and we are interested in a finite plan, we may use a dummy action with no effects, and require it to be executed just after the goal is fulfilled and not before (this can be done by suitably changing the goal). The dummy action acts as a marker for the end of the plan.

Formally, conformant planning can be described as follows: find an (infinite) sequence of actions $a_0 a_1 \cdots$ such that for any sequence $\sigma_0\sigma_1 \cdots$ of truth assignments to fluents, such that $a_0 a_1 \cdots$ and $\sigma_0\sigma_1 \cdots$ satisfy the system specification Γ, the goal γ is also satisfied.

Again, this condition cannot be expressed in LTL. However, it can be expressed in QLTL as follows. Let us introduce the formula

$$Plan(\Gamma, \gamma, \vec{a}) \doteq \forall \vec{f}.\Gamma(\vec{a}, \vec{f}) \rightarrow \gamma(\vec{a}, \vec{f})$$

which characterizes the sequences of actions such that, for all interpretations of the fluents consistent with the system specification, the goal is fulfilled. Hence, without considering legality, we can express plan existence as satisfiability of

$$Plan(\Gamma, \gamma, \vec{a})$$

This is a Π_1^{QLTL} formula. Considering that conformant planning is already EXPSPACE-hard for much simpler settings than the one considered here [18, 11] (where each action is always possible), we get:

Theorem 9 *Verifying existence of a (non necessarily legal) plan in* LTL *is EXPSPACE-complete.*

To verify existence of a legal plan, we simply have to check the satisfiability of

$$Plan(\Gamma, \gamma, \vec{a}) \wedge Legal(\Gamma, \vec{a})$$

Observe that, due to the presence of the legality check, this is a Π_2^{QLTL} formula. Hence, by Theorems 4 and 8, we get:

Theorem 10 *Verifying existence of a legal plan in* LTL *is 2-EXPSPACE-complete.*

To actually find a plan (either with legality check or not), one can develop an algorithm based on Büchi automata. Let us consider the case without legality check. Let $\varphi(\vec{a}, \vec{f})$ be $\Gamma(\vec{a}, \vec{f}) \rightarrow \gamma(\vec{a}, \vec{f})$. We have to check satisfiability of $\exists \vec{a}.\forall \vec{f}.\varphi(\vec{a}, \vec{f})$, i.e., $\exists \vec{a}.\neg\exists \vec{f}.\neg\varphi(\vec{a}, \vec{f})$ This can be reduced to checking nonemptiness of a Büchi automaton constructed as follows. We build an automaton A_1 corresponding to the LTL formula $\neg\varphi(\vec{a}, \vec{f})$ (exponential step). Then we project out $\vec{f}$ from such an automaton, obtaining an automaton A_2 accepting sequences of actions (polynomial step). Next we complement A_2 obtaining A_3 (exponential step), and check for nonemptiness of A_3. As mentioned, the nonemptiness algorithm for Büchi automata returns a witness for nonemptiness, which can be interpreted as a plan. The plan returned consists of two parts: a sequence arriving to a certain state, and a second sequence that forms a cycle back into that state. Thus, such plans have finite representations. The construction of the automaton is double exponential in both the size of the system specification Γ and of the goal specification γ. Observe that this algorithm is optimal, wrt complexity, for plan existence, since one can perform complementation and the final intersection on the fly, while checking for nonemptiness [40]. Using the same kind of automata manipulations we can build an automaton for checking existence of plans guaranteed to be legal.

6 CONCLUSIONS

In this paper, we have studied reasoning about action effects, legality of actions, and conformant planning in the setting of LTL. We have provided reasoning techniques based on a second-order extension of LTL.

The results obtained extend immediately to other variants of LTL such as μLTL, which is an extension of LTL with explicit fixpoint operators that is able to express any ω-regular property [37]. The key is that the reasoning techniques rely on an exponential translation of LTL to Büchi automata [39]. Such a translation is also known for μLTL. In addition, one may directly specify the system by a Büchi automaton instead of an LTL formula. Then, adopting the techniques discussed in this paper, it can be shown that reasoning on action effects remains PSPACE-complete, synthesizing non-necessarily legal plans remains EXPSPACE-complete, while testing legality becomes EXPSPACE-complete.

We have adopted a very general non-prescriptive notion of legality of actions, which turned out to be quite sophisticated and complex to verify. It would be very interesting to study such a notion in the branching-time setting at the base of situation calculus, dynamic logics, branching-time logics, etc. Note that the complexity is probably going to increase, by moving to the branching-time setting, which is in fact richer than the linear-time setting considered here. Also, one should stress that for system specifications of a special form, checking legality of sequences of actions may become much easier. For example, if our system only contains formulas expressing successor state axioms (see Section 3) then all actions are always legal.

Finally, algorithms for checking nonemptiness of Büchi automata, which are at the base of reasoning procedures for LTL and QLTL, have proved to be well suited for scaling up to very large systems. A breakthrough technology has been the use of *symbolic methods* and the experimental results on adopting symbolic techniques for planning under incomplete information are quite promising [7, 8, 5]. The symbolic techniques can be adapted to our framework. So, in spite of the high worst-case complexity, the scalability of the algorithms involved, hint that the automata-theoretic approach may actually be feasible, even in the general setting considered here.

Acknowledgements

The third author was supported in part by NSF grants CCR-9700061, CCR-9988322, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

[2] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2):123–191, 2000.

[3] C. Baral, M. Gelfond, and A. Provetti. Representing actions: laws, observations and hypotheses. *J. of Logic Programming*, 31(1–3):201–243, 1997.

[4] H. Barringer *et al.*MetateM: an introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.

[5] P. Bertoli, A. Cimatti, and M. Roveri. Heuristic search + symbolic model checking = efficient conformant planning. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, *pp.* 467–472, 2001.

[6] S. Cerrito and M. C. Mayer. Bounded model search in linear temporal logic and its application to planning. In *Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98)*, *LNAI* 1397, *pp.* 124–140. Springer, 1998.

[7] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking. In *Proc. of the 4th Eur. Conf. on Planning (ECP'97)*, 1997.

[8] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *J. of Artificial Intelligence Research*, 13:305–338, 2000.

[9] M. Daniele, P. Traverso, and M. Y. Vardi. Strong cyclic planning revisited. In *Proc. of the 5th Eur. Conf. on Planning (ECP'99)*, *LNAI* 1809, *pp.* 35–48. Springer, 1999.

[10] G. De Giacomo, L. Iocchi, D. Nardi, and R. Rosati. A theory and implementation of cognitive mobile robots. *J. of Logic and Computation*, 9(5):759–785, 1999.

[11] G. De Giacomo and M. Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Proc. of the 5th Eur. Conf. on Planning (ECP'99)*, *LNAI* 1809, *pp.* 226–238. Springer, 1999.

[12] P. Doherty, J. Gustafsson, L. Karlsson, and J. Kvarnström. (TAL) Temporal Action Logics: language specification and tutorial. *Elect. Trans. on Artificial Intelligence*, 2(3–4):273–306, 1998.

[13] E. A. Emerson. Automated temporal reasoning about reactive systems. In *Logics for Concurrency: Structure versus Automata*, *LNCS* 1043, *pp.* 41–101. Springer, 1996.

[14] O. Etzioni *et al.* An approach to planning with incomplete information. In *Proc. of the 3rd Int. Conf. on Knowledge Representation and Reasoning (KR'92)*, *pp.* 115–125, 1992.

[15] B. Finkbeiner. Language containment checking with nondeterministic BDDs. In *Proc. of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, *LNCS* 2031, *pp.* 24–38. Springer, 2001.

[16] M. Gelfond and V. Lifschitz. Action languages. *Elect. Trans. on Artificial Intelligence*, 3(16), 1998.

[17] M. R. Genesereth and I. R. Nourbakhsh. Time-saving tips for problem solving with incomplete information. In *Proc. of the 11th Nat. Conf. on Artificial Intelligence (AAAI'93), pp.* 724–730, 1993.

[18] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. of the 5th Eur. Conf. on Planning (ECP'99)*, 1999.

[19] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. on Computational Logic*, 2(3):408–429, 2001.

[20] R. P. Kurshan. *Computer Aided Verification of Coordinating Processes.* Princeton Univ. Press, 1994.

[21] J. Kvarnström and P. Doherty. TALplanner: a temporal logic based forward chaining planner. *Ann. of Mathematics and Artificial Intelligence*, 30:119–169, 2001.

[22] N. McCain and H. Turner. Satisfiability planning with causal theories. In *Proc. of the 6th Int. Conf. on Knowledge Representation and Reasoning (KR'98), pp.* 212–223, 1998.

[23] J. McCarthy. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39,171–172, 1980.

[24] R. Miller and M. Shanahan. The event calculus in classical logic — Alternative axiomatisations. *Elect. Trans. on Artificial Intelligence*, 4(16), 1999.

[25] M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), pp.* 479–484, 2001.

[26] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th Annual Symp. on the Foundations of Computer Science (FOCS'77), pp.* 46–57, 1977.

[27] R. Reiter. Narratives as programs. In *Proc. of the 7th Int. Conf. on Knowledge Representation and Reasoning (KR 2000), pp.* 99–108, 2000.

[28] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, 2001.

[29] E. Sandewall. *Features and Fluents: The Representation of Knowledge about Dynamical Systems.* Clarendon Press, Oxford, 1994.

[30] M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Law of Inertia.* The MIT Press, 1997.

[31] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *J. of the ACM*, 32(3):733–749, 1985.

[32] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[33] D. E. Smith and D. S. Weld. Conformant Graphplan. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98), pp.* 889–896, 1998.

[34] M. Thielscher. Causality and the qualification problem. In *Proc. of the 5th Int. Conf. on Knowledge Representation and Reasoning (KR'96), pp.* 51–62, 1996.

[35] M. Thielscher. Introduction to the fluent calculus. *Elect. Trans. on Artificial Intelligence*, 3(14), 1998.

[36] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, vol. B, *pp.* 133–192. Elsevier Science, 1990.

[37] M. Y. Vardi. A temporal fixpoint calculus. In *Proc. of the 15th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages (POPL'88), pp.* 250–259, 1988.

[38] M. Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *Proc. of the 7th Int. Conf. on Computer Aided Verification (CAV'95), LNCS* 939, *pp.* 267–292. Springer, 1995.

[39] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata, LNCS* 1043, *pp.* 238–266. Springer, 1996.

[40] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[41] D. S. Weld and O. Etzioni. The first law of robotics (a call to arms). In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI'94), pp.* 1042–1047, 1994.