LTL Goal Specifications Revisited

Andreas Bauer and **Patrik Haslum**[1]

Abstract. The language of linear temporal logic (LTL) has been proposed as a formalism for specifying temporally extended goals and search control constraints in planning. However, the semantics of LTL is defined wrt. *infinite* state sequences, while a finite plan generates only a finite trace. This necessitates the use of a finite trace semantics for LTL. A common approach is to evaluate LTL formulae on an infinite extension of the finite trace, obtained by infinitely repeating the last state. We study several aspects of this finite LTL semantics: we show its satisfiability problem is PSpace-complete (same as normal LTL), show that it complies with all equivalence laws that hold under standard (infinite) LTL semantics, and compare it with other finite trace semantics for LTL proposed in planning and in runtime verification. We also examine different mechanisms for determining whether or not a finite trace satisfies or violates an LTL formula, interpreted using the infinite extension semantics.

1 INTRODUCTION

The classical planning problem asks for a sequence of actions transforming an initial state into a state satisfying the goal specification. But for several reasons, there may be additional restrictions on the action sequence which are not easily captured by the goal state condition. These may reflect user requirements (often called "temporally extended goals") or may be intended as search control information, guiding the planner to sequences more profitable to explore.

The language of *linear temporal logic* (LTL) [14] has been proposed by several researchers as a suitable formalism for specifying such plan constraints, whatever the motivation for imposing them [1, 6, 2, 17]. The plan constraints available in the PDDL3 formalism [9] are also equivalent to a limited subset of LTL.

LTL formulae, when used in this manner, are evaluated over the sequence of states, or *trace*, generated by the action sequence. However, there is a significant mismatch between this use of LTL in planning and the semantics of the logic. The truth value of an LTL formula is defined wrt. an *infinite* state sequence, while a finite plan of course only generates a finite trace. This discrepancy can of course be resolved by generating infinite (cyclic) plans, as done by, e.g., Kabanza and Thiébaux [10], but the more common solution is to interpret LTL formulae according to some *finite trace semantics*.

The finite trace LTL interpretation most common in planning, since the work of Bacchus & Kabanza [1], is to view the trace generated by a finite plan as an infinite trace in which the last state repeats infinitely. We term this the *infinite extension semantics*, or IE-LTL. This is a reasonable interpretation, as it reflects the classical planning assumption that nothing other than the actions in the plan changes the world state. It is, however, by no means the only possibility. For instance, Baier and McIlraith [2] propose a different finite trace semantics (actually equivalent to the FLTL semantics; see section 4). PDDL3 plan constraints are also given an interpretation directly over finite traces (also equivalent to an IE-LTL specification).

Planning is not the only application area where LTL formulae over finite traces have been considered. As an example, runtime verification is an "on-line" form of model checking where properties, expressed in LTL, are checked incrementally against the finite trace generated by the running system (rather than against all possible infinite traces, as in normal "off-line" model checking; e.g, [5]). Several finite trace semantics for LTL, all different from the infinite extension semantics used in planning, have been proposed for use in runtime verification (cf. [3] for an overview).

The adoption of any finite trace semantics for LTL formulae has consequences. For example, there are LTL formulae that are only satisfiable by traces that infinitely alternate between two or more states[2]: a simple example is $\Box\Diamond p \wedge \Box\Diamond \neg p$. Clearly, such a formula can never be true in any trace where all but a finite prefix of states are the same. Hence the meaning of satisfiability changes in the finite setting.

We study several aspects of the infinite extension semantics for LTL in planning. We examine the relationship between IE-LTL and standard LTL semantics, as well as a number of other finite trace semantics, and show that satisfiability of IE-LTL is PSpace-complete. A practical use of this is that a formula representing a temporally extended goal can be tested for satisfiability *wrt. the correct semantics* prior to planning. The complexity of this check depends only on the size of the goal formula and on no other parts of the planning problem. We also show that under certain conditions, IE-satisfiability of a universally quantified formula can be decided without grounding.

Planners for temporally extended (LTL) goals need efficient mechanisms for evaluating the truth of a formula over a finite trace wrt. the chosen semantics. Methods that have been proposed include formula progression [1], and compiling extended goals into ordinary end-state goals by modifying the planning domain [6, 7, 2, 8]. In forward-chaining state space search, progression offers incremental evaluation and thus a mechanism for early pruning of action sequences that cannot lead to a valid plan. However, the size of the progressed formula can grow exponentially with the length of the action sequence. The alternative is to construct an automaton accepting those traces that are models of the goal formula, according to the IE-LTL semantics. This can be done by a modification of the construction of Büchi automata for standard LTL. The size of the automaton may be exponential, but once constructed it permits testing traces incrementally in time that is independent of the trace length, and a slightly different construction allows for efficient detection of dead ends.

[1] Australian National University, and NICTA. email: {andreas.bauer, patrik.haslum}@anu.edu.au. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. Patrik Haslum is supported by ARC project DP0985532 "Exploiting Structure in AI Planning".

[2] Note that even the extremely restricted constraint language of PDDL3 allows formulae of this kind to be specified [8].

2 BACKGROUND

Let AP be a non-empty set of atomic propositions.[3] A complete assignment of truth values to propositions in AP is a *state*. Let $S = 2^{AP}$ be the set of all states. S^* is the set of *finite* sequences of elements of S, including the empty (zero-length) sequence ϵ, $S^+ = S^* - \epsilon$ the set of non-empty finite sequences, and S^ω is the set of *infinite* sequences over S. For a state $s \in S$, s^* denotes a finite (possibly empty) sequence of repetitions of s, while s^ω denotes the infinite sequence of repetitions of s. As a notational convention, we use the letters w, w', etc. for infinite traces, and the letters u, v, u', etc. for finite traces.

For a finite trace $u = u_0 \dots u_n$, $\mathrm{last}(u)$ denotes the last state in u, i.e., u_n. The *infinite extension* of a finite trace u, is the infinite trace obtained by appending to u an infinite number of repetitions of $\mathrm{last}(u)$, i.e., $u\ \mathrm{last}(u)^\omega$. Note that there is no explicit symbol for trace concatenation. For any (infinite or finite) trace w, w^i denotes the suffix of w from position i onwards (inclusive, i.e., $w^0 = w$), whereas w_i denotes the ith element in w (i.e., $w_i \in S$).

The set of well-formed LTL formulae over vocabulary AP, LTL(AP), are given by the following grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \bigcirc\varphi$, where $p \in AP$ and $\varphi \in \mathrm{LTL}(AP)$. The truth of an LTL formula φ is defined wrt. an infinite trace $w \in S^\omega$ and $i \in \mathbb{N}^{\geq 0}$ as follows:

$$
\begin{array}{lcl}
w^i \models p \in AP & \Leftrightarrow & p \in w_i \\
w^i \models \neg\varphi & \Leftrightarrow & w^i \not\models \varphi \\
w^i \models \varphi_1 \vee \varphi_2 & \Leftrightarrow & w^i \models \varphi_1 \vee w^i \models \varphi_2 \\
w^i \models \bigcirc\varphi & \Leftrightarrow & w^{i+1} \models \varphi \\
w^i \models \varphi_1\,\mathcal{U}\,\varphi_2 & \Leftrightarrow & \exists k \geq i : ((w^k \models \varphi_2) \wedge \\
 & & \quad \forall l : (i \leq l < k \Rightarrow w^l \models \varphi_1))
\end{array}
$$

When $w^0 \models \varphi$ holds, we also write $w \models \varphi$. Besides the standard Boolean connectives, additional modal operators, such as the common *always* ($\Box$) and *eventually* ($\Diamond$), and *weak until* ($\mathcal{W}$) can be defined as abbreviations for formulae constructed using the basic LTL language: $\Diamond\varphi \equiv (\textsc{true}\,\mathcal{U}\,\varphi)$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$, and $\varphi\,\mathcal{W}\,\psi \equiv (\varphi\,\mathcal{U}\,\psi) \vee \Box\varphi$. As a dual to $\mathcal{U}$, the *release* operator $\varphi\,\mathcal{R}\,\psi$ is defined as $\neg(\neg\varphi\,\mathcal{U}\,\neg\psi)$.

For any formula $\varphi \in$ LTL, we can construct a non-deterministic Büchi automaton (NBA) $\mathcal{A} = (S, Q, Q_0, \delta, F)$, where S is the alphabet of states (over the vocabulary AP), Q the finite set of automaton states, $Q_0 \subseteq Q$ the initial states, δ the transition relation, and $F \subseteq Q$ the set of accepting states, such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$, i.e., the set of infinite traces accepted by $\mathcal{A}$ equals the set of models for φ (cf. [16]). The language of an NBA is determined by a labelling on either the transitions, in which case δ is defined as $Q \times S \to 2^Q$, or by a labelling function on states, $Q \to S$. The two types of automata are semantically equivalent. In either case, an infinite run through an NBA corresponds to an infinite sequence of labels (i.e., a trace), which is accepted by the NBA if the run contains infinitely often a state in F. The size of $\mathcal{A}$, measured as the number of automaton states, is, in the worst case, exponential in the size of φ for either type of automaton.

3 INFINITE EXTENSION SEMANTICS

In planning, the truth value of an LTL formula over a finite trace is often taken to be the truth value that the formula is given by the standard LTL semantics over the infinite trace constructed by appending an infinite repetition of the last state. We call this infinite trace the *infinite extension* of the finite trace, and this interpretation of LTL formulae the *infinite extension semantics*, or IE-LTL for short. It is formally defined as follows.

Definition 1 (Infinite Extension Semantics for LTL) *Let $\varphi \in$ LTL be a formula, and $u \in S^+$ be a non-empty finite trace. The truth value of φ in u according to the* infinite extension semantics*, denoted $[u \models_{IE} \varphi]$, is defined as*

$$
[u \models_{IE} \varphi] := \begin{cases} \top & \text{if } u\ \mathrm{last}(u)^\omega \models \varphi, \\ \bot & \text{otherwise.} \end{cases}
$$

Note that $[u \models_{IE} \varphi]$ is only defined for a *non-empty* finite trace u. For planning this is not an important restriction, since there is always an initial state. As a notational convention, we will be using brackets ("$[\cdot]$") around a satisfaction relation that is defined over finite traces, to differentiate from the standard relation over infinite traces.

Relationship Between IE-LTL and LTL The IE-LTL semantics is closely related to the standard, infinite trace LTL semantics. The difference is that in IE-LTL, we consider only infinite traces that have a certain structure, viz. that are composed of an arbitrary finite prefix followed by an infinite tail of identical states. Obviously, these are a strict subset of all possible infinite traces, and as a consequence, we have the following result.

Proposition 2 *Any equivalence that holds under normal (infinite) LTL semantics holds also under IE-LTL semantics.*

Proof: This is most easily seen by considering the contrapositive: Suppose φ and ψ are LTL formulae not equivalent under IE-LTL semantics. Then there is a finite trace, u, such that φ and ψ have different truth values in u, i.e., such that $[u \models_{IE} \varphi] \neq [u \models_{IE} \psi]$. This implies that the truth values of φ and ψ, according to standard LTL semantics, over the infinite trace $u\,\mathrm{last}(u)^\omega$, are also different. Thus, they cannot be equivalent under normal LTL semantics. □

The converse of Proposition 2 is not true: there are equivalences that hold under IE-LTL semantics but not under standard LTL semantics. As an example, consider the formulae (a) $\Diamond\Box p \vee \Diamond\Box q$ and (b) $\Diamond\Box(p \vee q)$. That (a) implies (b), under either standard LTL or IE-LTL semantics, is easy to see. The latter formula is satisfied by a trace that alternates infinitely between states where $p \wedge \neg q$ hold and states where $\neg p \wedge q$ hold, while the former is not; hence the two formulae are not equivalent, under standard LTL semantics. But such a trace cannot be the infinite extension of any finite trace. In any finite trace u such that $[u \models_{IE} \Diamond\Box(p \vee q)] = \top$, p or q, or both, must hold in the repeated state $\mathrm{last}(u)$, and thus at least one of $[u \models_{IE} \Diamond\Box p]$ and $[u \models_{IE} \Diamond\Box q]$ is true; the formulae are IE-equivalent.

4 ALTERNATIVE FINITE TRACE SEMANTICS

Finite trace semantics for LTL all have to deal with the same problem, namely how to interpret outstanding eventualities in a finite trace that does not allow for a conclusive answer. For example, is the formula $\bigcirc p$ true or false at the end of a finite trace? There are arguments for both choices, but whichever is made, the truth value can change when a new state is appended to the trace.

Next, we discuss a few different finite trace semantics, and their relationships to IE-LTL. These originate in the area of runtime verification (cf. [3]), though FLTL has been used also in planning.

[3] In planning, the propositional vocabulary is typically the set of all instantiations of a collection of predicates with objects from a finite set. Mostly, we will consider AP as a set of opaque symbols, but sometimes we'll also make use of the "lifted" view.

FLTL Semantics FLTL[4] extends LTL with a *weak next* operator, denoted $\overline{\bigcirc}\varphi$, with the intuitive meaning that if a next state exists, then it has to satisfy φ. Formally, the FLTL semantics of the two next operators is defined as follows:

$$[u \models_F \bigcirc\varphi] := \begin{cases} [u^1 \models_F \varphi] & \text{if } u^1 \neq \epsilon \\ \bot & \text{otherwise.} \end{cases}$$
$$[u \models_F \overline{\bigcirc}\varphi] := \begin{cases} [u^1 \models_F \varphi] & \text{if } u^1 \neq \epsilon \\ \top & \text{otherwise.} \end{cases}$$

This yields the following relationship between the two operators: $[u \models_F \neg(\bigcirc\varphi)] = [u \models_F \overline{\bigcirc}\neg\varphi]$. Remaining operators are interpreted as in standard LTL, but with respect to the finite trace. Thus, for the until operator, where $n = |u| > 0$, we have in FLTL:

$$[u \models_F \chi \,\mathcal{U}\, \psi] := \begin{cases} \top & \exists k \in \{0, \ldots, n-1\} : ([u^k \models_F \psi] = \top \wedge \\ & \quad \forall l : (0 \leq l < k \Rightarrow [u^l \models_F \chi] = \top)) \\ \bot & \text{otherwise.} \end{cases}$$

Baier and McIlraith [2] propose a finite trace semantics, which they call f-FOLTL, for planning. They interpret the next operator strongly, and instead of a weak next operator define a 0-ary operator FINAL, which is true only in the last state. However, it is easy to see that this semantics is equivalent to FLTL, since $\overline{\bigcirc}\varphi \equiv (\bigcirc\varphi) \vee \text{FINAL}$, and $\text{FINAL} \equiv \neg \bigcirc \text{TRUE}$. They also note that there are "several obvious discrepancies in the interpretation of LTL and f-FOLTL formulae" [2], but argue that the finite trace interpretation is adequate, or even better than standard LTL, for expressing temporally extended planning goals.

Relationship Between IE-LTL and FLTL The IE-LTL and FLTL semantics differ in their treatment of the next (and weak next) operator. To see how, consider the value of formulae $\bigcirc p$ and $\overline{\bigcirc} p$ on a trace u that consists of a single state (i.e., $|u| = 1$). In FLTL, $[u \models_F \bigcirc p] = \bot$, because there is no next state, while $[u \models_F \overline{\bigcirc} p] = \top$. The IE-LTL value, on the other hand, is decided by the value of p in the last (and only) state of u. If p is true in $\text{last}(u)$, $\bigcirc p$ will hold in $\text{last}(u)^\omega$, while if p does not hold in $\text{last}(u)$ it will not.

This discrepancy extends to satisfiability. For example, Baier and McIlraith [2] note that the formula $\Box(\varphi \rightarrow \bigcirc\psi) \wedge \Box(\psi \rightarrow \bigcirc\varphi)$ is in FLTL equivalent to $\Box(\neg\varphi \wedge \neg\psi)$, since if either φ or ψ is ever true, the first formula would require an infinite trace to be true. In IE-LTL, however, this equivalence does not hold, since the first formula is also satisfied by, e.g., any single state trace where $\varphi \wedge \psi$ holds.

However, as long as we avoid using next (and weak next), IE-LTL and FLTL agree, as the next proposition shows.

Proposition 3 *Let φ be a LTL formula without $\bigcirc$ or $\overline{\bigcirc}$ operators, and $u \in \Sigma^+$ a finite trace: $[u \models_{IE} \varphi] = [u \models_F \varphi]$.*

Proof: By structural induction. The interpretation of Boolean connectives is the same in both semantics, so the only interesting case is the until operator. Let $\varphi = \chi \,\mathcal{U}\, \psi$, and let $|u| = n \geq 1$.

If $[u \models_F \chi \,\mathcal{U}\, \psi] = \top$, then $\exists 0 \leq k \leq n-1$. $[u^k \models_F \psi] = \top$ and $\forall 0 \leq l < k$. $[u^l \models_F \chi] = \top$. But then $uw \models \chi \,\mathcal{U}\, \psi$, for any $w \in S^\omega$, including $u\,\text{last}(u)^\omega$. Thus, $[u \models_{IE} \chi \,\mathcal{U}\, \psi] = \top$.

If $[u \models_{IE} \chi \,\mathcal{U}\, \psi] = \top$, then $w = u\,\text{last}(u)^\omega \models \chi \,\mathcal{U}\, \psi$, which implies $\exists 1 \leq k$. $w^k \models \psi$ and $\forall 1 \leq l < k$. $w^l \models \chi$. The least such k cannot be greater than $n-1$, since every suffix from w^{n-1} and onwards is identical. Thus, there exists a $k \leq n-1$ such that $[u^k \models_{IE} \psi] = \top$ and $\forall 0 \leq l < k$. $[u^l \models_{IE} \chi] = \top$, which by inductive assumption implies that $[u^k \models_F \psi] = \top$ and $\forall 0 \leq l < k$. $[u^l \models_F \chi] = \top$. Thus, $[u \models_F \varphi] = \top$. □

LTL$_3$ and RV-LTL The LTL$_3$ semantics maps LTL formulae over finite traces to three possible values: $\top$, $\bot$, and "inconclusive", denoted by ?. The first two values are taken by $[u \models_3 \varphi]$ when the value of φ, according to standard LTL semantics, is the same no matter what extensions follows after u. In other words,

$$[u \models_3 \varphi] := \begin{cases} \top & \text{if } \forall w \in S^\omega.\ uw \models \varphi \\ \bot & \text{if } \forall w \in S^\omega.\ uw \not\models \varphi \\ ? & \text{otherwise.} \end{cases}$$

Thus, instead of evaluating $\bigcirc\varphi$ prematurely to either $\top$ or $\bot$, a third value denotes that the current trace is inconclusive.

The RV-LTL semantics further discriminates inconclusive situations using four different truth values. It combines LTL$_3$ and FLTL, in the sense that for traces such that $[u \models_3 \varphi]$ equals $\top$ or $\bot$, RV-LTL assigns the same value, while for traces where $[u \models_3 \varphi] = ?$, RV-LTL uses the FLTL interpretation to determine whether the value is "possibly true" ($\top^p$), or "possibly false" ($\bot^p$).

Corollary 4 *Let φ be a LTL formula without $\bigcirc$ and $\overline{\bigcirc}$ operators, and $u \in \Sigma^+$ a finite trace. It then holds that $[u \models_{RV} \varphi] \in \{\top, \top^p\}$ iff $[u \models_{IE} \varphi] = \top$, and, equivalently, that $[u \models_{RV} \varphi] \in \{\bot, \bot^p\}$ iff $[u \models_{IE} \varphi] = \bot$.*

Proof: When $[u \models_{RV} \varphi] \in \{\top, \bot\}$, the value is given by the LTL$_3$ semantics, i.e., either $uw \models \varphi$ for every possible continuation $w \in S^\omega$, or $uw \not\models \varphi$ for every $w \in S^\omega$. Since $\text{last}(u)^\omega$ is one such possible continuation, IE-LTL assigns the same value. When $[u \models_{RV} \varphi] \in \{\top^p, \bot^p\}$ the value is given by the FLTL semantics, and the results follows from Proposition 3. □

5 PLANNING WITH TEMPORALLY EXTENDED GOALS

We adopt a standard definition of classical planning. A propositional *planning domain* consists of a set of atomic propositions, AP, and a set A of actions. As mentioned, AP is typically obtained by grounding out the domain predicates with a finite set of objects. Each action $a \in A$ is described by a precondition, $\text{pre}(a)$, which is a Boolean formula over AP, and an effect, which is a (partial) function $\text{eff}(a) : S \longrightarrow S$, such that $\text{eff}(a)(s)$ is defined for any $s \in S$ such that $\text{pre}(a)$ holds in s. The precise definition of the mapping depends on the action formalism, and is not important for our purposes.

A *planning problem* consists of a domain, an initial state $s_0 \in S$ and a goal, G, which is expressed as an LTL formula. If we wish to state an ordinary reachability goal, this can be expressed as $\Diamond\Box\alpha$, where α is the goal state condition.

A finite sequence of actions $a_1, \ldots, a_n$ generates a finite trace, $u_0 u_1 \ldots u_n \in S^*$, where u_0 is the initial state of the planning problem, and $u_i = \text{eff}(a_i)(u_{i-1})$ for $i \leq n$. The action sequence is *executable* iff $\text{pre}(a_i)$ holds in u_{i-1} for $i = 1, \ldots, n$. Note that if the action sequence is executable, the trace is well-defined.

An executable action sequence is a solution to the planning problem (a *plan*) iff the goal formula G is true in the finite trace generated by the sequence, *according to the chosen finite trace semantics*.

[4] FLTL corresponds to a variant of LTL over finite traces that was originally introduced by Manna and Pnueli [13]. The name FLTL, short for "finite LTL," was later used to refer to this logic by Bauer et al. [3].

This is often taken to be the infinite extension semantics, and there are good reasons for this choice: this interpretation follows naturally from the classical planning assumption that nothing other than the actions in the plan changes the world state, and, as we have shown, all equivalences that hold in standard LTL also hold in IE-LTL, so operations such as rewriting a formula to negation normal form can be done as usual. However, nothing prevents us from choosing a different finite trace interpretation. As noted earlier, Baier and McIlraith [2] assume the FLTL semantics. Alternatively, we could adopt the LTL_3 semantics, and require that for a plan to be valid, the goal formula should evaluate to $\top$. This is a stricter condition for plan validity, accepting only plans that guarantee the temporally extended goal is satisfied no matter what happens after the end of the plan.

Modelling Temporally Extended Goals To give some examples of temporally extended goals expressed in LTL, we consider the Miconic domain[5], which models planning for a bank of elevators equiped with "destination control" [11]. The goal is to serve each passenger (i.e., deliver them to their destination floor),

$$\forall \texttt{?p} : \texttt{passenger}\, \Diamond\Box \texttt{served(?p)},$$

but there are often also other constraints on plans in this domain [11]. For example, the formula

$$\begin{array}{l}\forall \texttt{?p} : \texttt{passenger}\,(\texttt{above(destin(?p), origin(?p))} \rightarrow \\ \quad \forall \texttt{?e} : \texttt{elevator}\,\Box(\texttt{boarded(?p, ?e)} \rightarrow \\ \quad\quad \forall \texttt{?f} : \texttt{floor} \\ \quad\quad\quad ((\texttt{at(?e, ?f)} \rightarrow \texttt{above(?f, origin(?p))}) \\ \quad\quad\quad \mathcal{U}\ \texttt{served(?p)}))))\end{array}$$

expresses that any passenger `?p` whose destination is above his origin floor, travels only upwards after boarding an elevator, while

$$\begin{array}{l}\forall \texttt{?p1, ?p2} : \texttt{passenger, ?e1, ?e2} : \texttt{elevator} \\ \quad \Box((\texttt{vip(?p1)} \wedge \neg \texttt{vip(?p2)} \wedge \\ \quad\quad \texttt{origin(?p1)} = \texttt{origin(?p2)}) \rightarrow \\ \quad\quad \neg(\texttt{boarded(?p2, ?e2)} \wedge \neg \texttt{boarded(?p1, ?e1)}))\end{array}$$

expresses that VIP passenger (`?p1`) should never be left waiting while a non-VIP passenger (`?p2`) is picked up from the same floor.

PDDL3 PDDL3 [8] extends earlier versions of the Planning Domain Definition Language (PDDL) in a number of ways, one of which is the introduction of *plan constraints* which express restrictions on the trace generated by the plan. The semantics of the PDDL3 plan constraint operators is defined directly in terms of the finite trace generated by a plan. However, their meaning is identical to that of the LTL formulae shown in Table 1, when those formulae are interpreted according to the infinite extension semantics, though note that for the "temporal" operators (lower part of the table), this equivalence holds only in the the unit time-step case; the semantics of these operators in the metric time case cannot be expressed in (untimed) LTL.

PDDL3 does not allow nesting of modal operators. Because of the limited form that plan constraints in PDDL3 may take, they can be compiled away with only polynomial increase in the size of the planning domain [8]. In spite of this, PDDL3 can express plan constraints that are satisfiable but not IE-satisfiable. A simple example is

```
(and (sometime-after p (not p))
     (sometime-after (not p) p)).
```

[5] The examples are based on the IPC2 STRIPS formulation of the domain. For improved readability, we use some function terms, like `destin(?p)`, which would in plain STRIPS need to be written using extra quantifiers, i.e., $\forall$`?fd : floor(destin(?p, ?fd)` $\rightarrow \ldots$).

PDDL3	IE-LTL
`(always` α`)`	$\Box\alpha$
`(sometime` α`)`	$\Diamond\alpha$
`(at-most-once` α`)`	$\Box(\alpha \rightarrow (\alpha\, \mathcal{W}\, \Box\neg\alpha))$
`(sometime-after` α β`)`	$\Box(\alpha \rightarrow \Diamond\beta)$
`(sometime-before` α β`)`	$(\neg\alpha\, \mathcal{W}\, \neg\alpha \wedge \beta)$
`(within` n α`)`	$\bigvee_{0\leq i\leq n} \bigcirc^i \alpha$
`(always-within` n α β`)`	$\Box(\alpha \rightarrow \bigvee_{0\leq i\leq n} \bigcirc^i \beta)$
`(hold-during` n m α`)`	$\bigwedge_{n\leq i<m} \bigcirc^i \alpha$
`(hold-after` n α`)`	$\bigcirc^n \Diamond \alpha$

Table 1. PDDL3 plan constraints and corresponding IE-LTL formulae. α and β are state formulae; n and m are non-negative integers. The expression $\bigcirc^n$ stands for n applications of the $\bigcirc$ operator.

6 CHECKING IE-LTL SATISFIABILITY

Next, we consider the problem of deciding satisfiability of an LTL formula wrt. infinite extension semantics, or IE-satisfiability for short. We show that its complexity is the same as that of standard LTL satisfiability. As noted earlier, the set of infinite traces obtained via infinite repetition of the last state of a finite trace is a strict subset of the set of all infinite traces. Thus,

Corollary 5 *IE-satisfiability implies satisfiability wrt. the standard LTL semantics.*

Theorem 6 *IE-satisfiability is PSpace-complete.*

Proof: Membership in PSpace can be shown by embedding IE-LTL into standard LTL. Let $\varphi \in$ LTL and AP be the set of propositions as used in φ. We construct in polynomial time a formula $\psi := \phi \wedge \Diamond \bigwedge_{a\in AP}((a \rightarrow \Box a) \wedge (\neg a \rightarrow \Box\neg a))$. Intuitively, ψ encodes that φ must be true and that from a certain point onwards, all propositions must retain their truth values forever. φ is IE-satisfiable iff ψ is IE-satisfiable, since the right-hand part is a tautology in all IE-LTL models. ψ is IE-satisfiable iff ψ is satisfiable in regular LTL, since the right-hand part ensures that the only possible LTL models are those models which are also considered in IE-LTL. Since LTL satisfiability is PSpace-complete, membership in PSpace follows.

We show hardness via a reduction from the propositional STRIPS planning problem, which is PSpace-hard [4]. Given a planning problem P with actions A, initial state s_0 and goal G, we construct an LTL formula φ_P such thatφ_P is IE-satisfiable iff there exists a plan for P. Note that G here is an ordinary end state goal.

The construction of φ_P is similar to an encoding of (bounded) planning into propositional logic, but using the next operator ($\bigcirc$) to link each state ("layer") to the following instead of indexing.

$$\begin{array}{rl}\varphi_P \equiv & \left(\bigwedge_{a\in A} a \rightarrow \text{pre}(a)\right) \wedge \left(\bigwedge_{a,a'\in A, a\neq a'} a \rightarrow \neg a'\right) \wedge \\ & \left(\bigwedge_{p\in AP} \varphi_{\text{next}(p)}\right) \wedge \varphi_{\text{init}} \wedge \Diamond\Box G.\end{array}$$

The first and second conjuncts ensure that at most one action is applied in each state and that this action is applicable. φ_{init} is the usual encoding of the initial state, i.e., the conjunction of literals true in s_0, and $\varphi_{\text{next}(p)}$ encodes the explanatory frame axiom for p: $\bigcirc p \leftrightarrow (\bigvee_{a\in A: p\in\text{add}(a)} a) \vee (p \wedge \bigwedge_{a\in A: p\in\text{del}(a)} \neg a)$. If there exists a plan for P, then the infinite extension of the trace it generates is a model of φ_P, and of required form; hence, φ_P is IE-satisfiable. If there is no plan for P, φ_P is unsatisfiable, and therefore also IE-unsatisfiable. Finally, the size of φ_P is at most quadratic wrt. P. □

The decision procedure for IE-satisfiability has exponential time complexity, but only in the size of the goal formula: it does not depend on any other part of the planning problem. However, temporally extended goal specifications often require something to hold for all objects of a certain type, in which case the grounded formula grows with the size of the problem. Fortunately, it is sometimes possible to check the IE-satisfiability of a universally quantified formula without grounding it. The next theorem shows one sufficient condition.

Theorem 7 *Let $\varphi = \forall x_1 \in D_1, \ldots, \forall x_n \in D_n.\ \psi(x_1, \ldots, x_n)$, where $\psi(x_1, \ldots, x_n)$ is a first-order LTL formula without quantifiers or functions, and let $\psi(d_1, \ldots, d_n)$ be an arbitrary instantiation of $\psi(x_1, \ldots, x_n)$ with objects $d_1 \in D_1, \ldots, d_n \in D_n$. If the domains $D_1, \ldots, D_n$ are all disjoint, then φ is IE-satisfiable iff $\psi(d_1, \ldots, d_n)$ is.*

Proof: If $\psi(d_1, \ldots, d_n)$ is IE-satisfiable, there exists a finite trace $u = u_0, \ldots u_n$ such that $[u \models_{IE} \psi(d_1, \ldots, d_n)] = \top$. From u we construct a finite trace u' satisfying φ. Let u_k be a state in u: the corresponding state u'_k in u' is defined as follows: Let P be an m-ary predicate, and $P(x_{i_i}, \ldots, x_{i_k})$ an occurrence P in φ, with argument terms $x_{i_i}, \ldots, x_{i_k}$. In $\psi(d_1, \ldots, d_n)$, there is a corresponding instantiation $P(d_{i_i}, \ldots, d_{i_m})$ of P. Let Γ be the set of all instantiations of $P(x_{i_i}, \ldots, x_{i_k})$, i.e., instantiations of P with first argument in the domain of x_{i_1}, second argument in the domain of x_{i_2}, etc., and let each instance in Γ have in u'_k the truth value that $P(d_{i_i}, \ldots, d_{i_m})$ has in u_k. Since variables range over disjoint domains, no occurrence of P in φ with different argument terms can instantiate to any of the elements of Γ. Thus this assignment is consistent.

That $[u' \models_{IE} \varphi] = \top$ follows directly from the construction. For any instantiation of the quantified variables $x_1, \ldots, x_n$, the truth values of all atomic propositions in the formula in every state are the same as in the corresponding state of the trace that satisfies $\psi(d_1, \ldots, d_n)$, thus making $\psi(x_1, \ldots, x_n)$ true. If $\psi(d_1, \ldots, d_n)$ is not IE-satisfiable then neither is φ, since for φ to be IE-satisfiable there must be a finite trace satisfying $\psi(x_1, \ldots, x_n)$ for every instantiation of $x_1, \ldots, x_n$, including $d_1, \ldots, d_n$. □

To illustrate the use of Thm. 7, consider the last example formula from the Miconic domain given in section 5: to check IE-satisfiability of this formula, we would have to ground `?e1`, `?e2`, and the implicit quantified `floor` variables, but not `?p1` and `?p2`, since the domains of VIP and non-VIP passengers are disjoint.

7 INCREMENTAL EVALUATION FOR SEARCH CONTROL

A planner seeking to achieve a temporally extended goal has to decide (1) when a plan satisfies a goal formula (goal test) and (2) when a partial plan cannot be extended to one that does (dead-end test).

In a forward-chaining, state space search-based planner, both tests are done on a large number of traces of increasing length, so it is clearly desirable that the test mechanism is efficient, and that its complexity does not grow with the length of the trace. Moreover, trading some extra effort prior to search, on preprocessing or optimising for faster per-state evaluation, is likely to pay off in this setting.[6]

[6] As an example, TALplanner, a forward-chaining planner that uses a temporal logic for search control, performs a sophisticated analysis of the control formulae prior to search with the aim of speeding up on-line formula evaluation, and this is one important reason for its good performance [12].

7.1 Formula Progression

Several planners have used formula progression to perform either goal or dead-end testing, or both [1, 10]. Progression splits the goal formula into a condition on the current state and a new goal that has to be achieved by the future trace: if the new goal is TRUE, it is a goal state; if the new goal is FALSE, it is a dead-end.

Progression is done by the inductive function $prog : \text{LTL} \times S \rightarrow \text{LTL}$, taking a formula φ and a state s as input and returning a new formula, such that $s\,w \models \varphi$ iff $w \models prog(\varphi, s)$, for any $w \in S^\omega$. For example, $prog(\Box\psi, s) = prog(\psi, s) \wedge \Box\psi$, where $prog(\psi, s)$ may yield a truth value immediately or after expanding to a more complicated formula. Other operators are handled similarly.

At worst, each progression step may double the size of the formula. Thus, without continuous simplification of formulae, the complexity of *each* progression step over an expanding trace may be exponential in the trace length. Rosu and Havelund [15, Thm. 2] have shown that this exponential growth is unavoidable: there exists a family of formulae φ, of increasing size, such that for some trace $u = u_0 \ldots u_n \in S^+$, *any* formula equivalent to $\text{prog}(\ldots(\text{prog}(\varphi, u_0)\ldots, u_n)$ is at least exponential in $|\varphi|$.

Thus, even if the result of each progression step is simplified, one may still have to perform an exponential amount of work for every new state that is added to the trace. Moreover, to ensure it remains single exponential, simplification must test formulae for equivalence, which in itself is a PSpace-complete problem. Thus, besides having already exponentially sized formulae, the simplification is also likely to require exponential time. These operations must then be carried out for every new state s that is added to the trace.

7.2 Length Independent Goal and Dead-End Tests

By way of the embedding described in Thm. 6 we can obtain an NBA for a formula φ that accepts only traces that are infinite extensions of a finite prefix by repeating the last state infinitely often. From this NBA we can derive an ordinary finite automaton which accepts exactly those finite prefixes. This automaton can be used directly to perform goal tests on traces of increasing length in constant time, or it can be compiled into the planning problem in various ways [6, 7, 2, 8]. An advantage of compilation is that this enables planning heuristics to guide search towards an accepting state. For dead-end detection, the compilation approach relies solely on the planner's inherent ability to detect states from which no goal state is reachable. Kabanza & Thiébaux [10], who consider infinite plans, convert the temporally extended goal into an NBA, from which a series of state-reachability goals is derived, but still use progression of search control formulae (wrt. the state goals) to detect dead-ends.

As an alternative, a slight modification of the construction yields an automaton which accepts traces such that the formula holds in *any* extension, not just ones formed by infinite repetition of the last state: by applying this construction to the *negation* of the goal formula, we obtain a dead-end test device. In the worst case, the automata may be of size exponential in that of the formula, but, compared to progression, they have the advantage that the processing of each new state in an expanding trace requires only constant time. In the remainder of this section, we sketch the details of the two automata constructions.

Let $\langle \mathcal{A}_\varphi = (S, Q, Q_0, \delta, F), \lambda \rangle$ be a state-labelled NBA accepting the models of the embedding of φ, i.e., $\varphi \wedge \Diamond \bigwedge_{a \in AP}((a \rightarrow \Box a) \wedge (\neg a \rightarrow \Box\neg a))$, and $\lambda : Q \rightarrow S$ its state-labelling function. Further, let for any $q \in Q$, $\mathcal{A}_\varphi(\{q\})$ be like $\mathcal{A}_\varphi$ except that its set of initial states is $\{q\}$ instead of Q_0. Whenever $\langle \mathcal{A}_\varphi, \lambda \rangle$ ac-

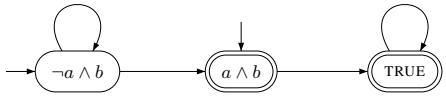

Figure 1. Finite automaton for $\varphi \equiv a\,\mathcal{R}\,b \wedge \Diamond a$. Initial states are marked with an incoming arrow, and accepting states with double outlines.

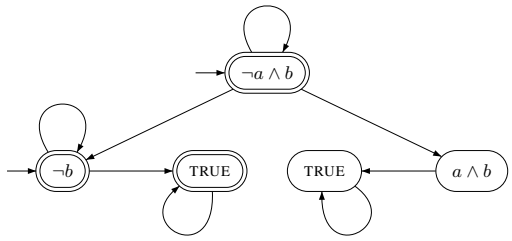

Figure 2. The state-labelled NBA for $\neg\varphi \equiv (\neg a\,\mathcal{U}\,\neg b) \vee \Box\neg a$.

cepts a trace $w = u\,\mathrm{last}(u)^\omega$, there exists a path through the states of $\mathcal{A}_\varphi$, consisting of a finite prefix, and a cycle with at least one accepting state in it. Using $\langle \mathcal{A}_\varphi, \lambda \rangle$, the finite automaton is constructed as follows: Check for all $q \in Q$ whether or not they are part of a (non-trivial) strongly connected component (SCC) which contains at least one accepting state, or which unavoidably leads to a state q that forms part of a universal SCC, i.e., $\mathcal{L}(\mathcal{A}_\varphi(\{q\})) = S^\omega$. Note that for any prefix $u = u_0 \ldots u_n$ of an accepted word of $\mathcal{A}_\varphi$ and any position $0 \leq i < n$, it holds that $u_i \models \lambda(q_i)$ and there exists a state $q_{i+1} \in \delta(q_i)$. Due to the embedding of φ, the NBA ensures that if q_{i+1} is part of an accepting SCC, then the infinite repetition of u_i will always yield a successor in the same SCC, and consequently a state in F, i.e., is an accepted word of the NBA. Hence, for every accepting SCC we define a set $\hat{F} \subseteq Q$ containing all its states as well as those leading unavoidably into universal SCCs. Let $\mathcal{F}$ be the union of all these sets. It is easy to see that the accepted language of the finite automaton $\hat{\mathcal{A}}_\varphi$, $\mathcal{L}(\hat{\mathcal{A}}_\varphi) \subseteq S^+$, consists of exactly those $u \in S^+$ for which $[u \models_{IE} \varphi] = \top$ holds. An example is given in Fig. 1, which depicts the finite automaton for the goal formula $\varphi \equiv a\,\mathcal{R}\,b \wedge \Diamond a$. Note that since $\hat{\mathcal{A}}_\varphi$ is an ordinary finite automaton (unlike $\mathcal{A}_\varphi$ which is an NBA) it can be determinised and minimised using standard algorithms. By making $\hat{\mathcal{A}}_\varphi$ deterministic, membership in $\mathcal{L}(\hat{\mathcal{A}}_\varphi)$ can be checked in constant time for each new state in the trace.

$\hat{\mathcal{A}}_\varphi$ performs only the goal test, i.e., it decides if the trace u generated by the current (partial) plan satisfies the goal. When that is not the case, it does not tell us if u can be extended to a valid plan or not. To perform the dead-end test, we can construct a second automaton, $\hat{\mathcal{A}}_{\neg\varphi}$, and then run the two in parallel. The construction of the dead-end monitor is slightly different, because it serves a different purpose: for the goal test, we only need to know if $[u \models_{IE} \varphi] = \top$, whereas for the dead-end test, we want to know if $[u\,v \models_{IE} \neg\varphi] = \top$ for *any* finite extension v of u. Therefore, the dead-end monitor can accept a trace only when it reaches an accepting SCC *and* every transition out of this SCC leads to a universal SCC. Consequently, for the state-labelled NBA, generated from $\neg\varphi$, we first have to check if there exists a state $q \in F$, which does *not* lead to a universal SCC, and eliminate it from F. To see a case where this occurs, consider the NBA for $\neg(a\,\mathcal{R}\,b \wedge \Diamond a)$, shown in Fig. 2: the state labelled by $\neg a \wedge b$ must be made non-accepting since it is possible to take a transition to a non-accepting state (i.e., one where the goal is true), by appending a state satisfying $a \wedge b$ to the trace.

8 SUMMARY

We have presented a formal account of IE-LTL, the finite trace semantics for LTL widely adopted in planning when LTL is used to express temporally extended goals, and shown that its satisfiability problem is PSpace-complete, as well as several results about its relation to standard LTL and other finite trace LTL semantics.

We also discussed the merits of mechanisms for performing goal and dead-end tests in a forward state space search for plans achieving an LTL goal. In contrast to the widely used formula progression method, which may from suffer an exponential growth in complexity as the length of the trace increases, methods based on automata can incrementally check the status of a trace in constant time for each new state added. Moreover, handling goal tests (and heuristic guidance towards the goal) and dead-end tests by separate devices may be better than using only one method.

REFERENCES

[1] F. Bacchus and F. Kabanza, 'Using temporal logic to control search in a forward chaining planner', in *Proc. 3rd European Worksh. on Planning (EWSP'95)*, (1995).

[2] J. Baier and S. McIlraith, 'Planning with first-order temporally extended goals using heuristic search', in *Proc. 21st Nntl. Conf. on AI (AAAI'06)*, (2006).

[3] A. Bauer, M. Leucker, and C. Schallhart, 'Comparing LTL semantics for runtime verification', *Journal of Logic and Computation*, (2009). In print. Pre-print available online.

[4] T. Bylander, 'Complexity results for planning', in *Proc. 12th Intl. Joint Conf. on Artif. Int. (IJCAI'91)*, pp. 274–279, (1991).

[5] S. Colin and L. Mariani, 'Run-time verification', in *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*, pp. 525–555, (2004).

[6] S. Cresswell and A. Coddington, 'Compilation of LTL goal formulas into PDDL', in *Proc. of the 15th European Conf. on Artif. Int., (ECAI'04)*, (2004).

[7] S. Edelkamp, 'On the compilation of plan constraints and preferences', in *Proc. of the 16th Intl. Conf. on Automated Planning and Scheduling (ICAPS'06)*, pp. 374–377, (2006).

[8] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, 'Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners', *Artif. Int.*, **173**(5-6), 619–668, (2008).

[9] A. Gerevini and D. Long, 'Plan constraints and preferences in PDDL3', Report RT 2005-08-47, University of Brescia, Italy, (2005).

[10] F. Kabanza and S. Thiébaux, 'Search control in planning for temporally extended goals', in *Proc. of the 15th Intl. Conf. on Automated Planning and Scheduling (ICAPS'05)*, pp. 130–139, (2005).

[11] J. Koehler and K. Schuster, 'Elevator control as a planning problem', in *Proc. 5th Intl. Conf. on Artif. Int. Planning and Scheduling (AIPS'00)*, pp. 331–338, (2000).

[12] J. Kvarnström, 'Applying domain analysis techniques for domain-dependent control in TALplanner', in *Proc. 6th Intl. Conf. on Artif. Int. Planning and Scheduling (AIPS'02)*, pp. 101–110, (2002).

[13] Z. Manna and A. Pnuelli, *Temporal Verification of Reactive Systems: Safety*, Springer, 1995.

[14] A. Pnueli, 'The temporal logic of programs', in *Proc. of the 18th Symp. on Found. of Comp. Sci. (FOCS'77)*, pp. 46–57, (1977).

[15] G. Rosu and K. Havelund, 'Rewriting-based techniques for runtime verification.', *Automated Software Engineering*, **12**(2), 151–197, (2005).

[16] K. Rozier and M. Vardi, 'LTL satisfiability checking', in *Proc. 14th Intl. SPIN Worksh. (SPIN'07)*, volume 4595 of *LNCS*, pp. 149–167, (2007).

[17] S. Thiébaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza, 'Decision-theoretic planning with non-markovian rewards', *Journal of AI Research*, **25**, 17–74, (2006).