Planning with First-Order Temporally Extended Goals Using Heuristic Search

Jorge A. Baier and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
Toronto, Canada
{jabaier,sheila}@cs.toronto.edu

Abstract

Temporally extended goals (TEGs) refer to properties that must hold over intermediate and/or final states of a plan. The problem of planning with TEGs is of renewed interest because it is at the core of planning with temporal preferences. Currently, the fastest domain-independent classical planners employ some kind of heuristic search. However, existing planners for TEGs are not heuristic and are only able to prune the search space by progressing the TEG. In this paper we propose a method for planning with TEGs using heuristic search. We represent TEGs using a rich and compelling subset of a first-order linear temporal logic. We translate a planning problem with TEGs to a classical planning problem. With this translation in hand, we exploit heuristic search to determine a plan. Our translation relies on the construction of a parameterized nondeterministic finite automaton for the TEG. We have proven the correctness of our algorithm and analyzed the complexity of the resulting representation. The translator is fully implemented and available. Our approach consistently outperforms TLPLAN on standard benchmark domains, often by orders of magnitude.

1 Introduction

In this paper we address the problem of planning with temporally extended goals (TEGs). TEGs are goals that refer to properties that must be achieved at various states along the execution of a plan, not just in the final state. They are compelling because they encode many realistic but complex goals that involve properties other than those concerning the final state. Examples include achieving several goals in succession (e.g., deliver all the priority packages and then deliver the regular mail), safety goals such as maintenance of a property (e.g., always maintain at least 1/4 tank of fuel in the truck), and achieving a goal within some number of steps (e.g., the truck must refuel at most 3 states after its final delivery). The problem of planning with TEGs is of renewed interest because it is at the core of planning with temporal preferences. Indeed, most preferences in PDDL3 [11]—the plan domain description language used in the 2006 International Planning Competition (IPC)— can be specified in linear temporal logic (LTL) [1].

[1] All of them can be expressed in LTL, in the absence of durative actions (i.e. when all actions are instantaneous).

TLPLAN [2] and TALPLAN [15] are examples of planners that can plan with TEGs. They do so by treating TEGs as temporal domain control knowledge (TDCK). TLPLAN's search strategy is one of blind search on a search space that is constantly pruned by the progression of the TDCK. This works well for safety-oriented TDCK (e.g., $\Box open(door)$) because TLPLAN prunes those actions that falsify the TDCK. Unfortunately, TLPLAN does not guide the search *towards* goals so it is less effective with respect to liveness properties such as $\Diamond at(Robot, Home)$ or other goals that have to be actively achieved.

Heuristic planners such as FF [13] are among the fastest planners for classical planning. Given a planning problem P, they evaluate the quality of states in the search space by computing a heuristic function over a relaxation of P. A common relaxation corresponds to disregarding the actions' negative effects. To compute the heuristic function a Graphplan-style planning graph [5] is expanded until all the goal facts appear in a fact node. To use techniques such as this directly with a TEG requires us to be able to determine which facts of the domain are true iff the TEG is true, thus enabling us to stop the graph expansion and to compute the heuristic. In this way, we could use the same classical heuristic search techniques for planning with TEGs.

In this paper we propose a method for performing heuristic search on planning problems with TEGs by exploiting the relationship between temporal logic and automata. Given a planning problem for a TEG, we transform it into a classical planning problem and apply a domain-independent heuristic search planner to actively guide search towards the goal. This new augmented domain, contains additional predicates that allow us to describe the (level of) achievement of the TEG. In particular, in this new domain there is a *classical goal* that is satisfied iff the TEG of the original problem is achieved.

To convert a TEG into a classical planning problem we provide a translation of f-FOLTL formulae to parameterized nondeterministic finite automata (PNFA). f-FOLTL is a version of LTL which we modify to include first-order (FO) quantifiers and to be interpreted only by finite computations. We have proven the correctness of our translation algorithm. Once the TEG is represented as an PNFA, we provide a construction of the classical planning problem using derived predicates (axioms). We analyze the space complexity of our construction, briefly discuss its superiority to ADL-operator based approaches, and propose techniques to reduce its size.

Our translators are fully implemented and are posted on the Web [2], together with our test suites. They output PDDL problem descriptions, making our approach amenable to use with a variety of classical planners. We have experimented with FF's extension for derived predicates, FF$_{\mathcal{X}}$ [21], over 3 benchmark domains, comparing our results to TLPLAN. Our method is consistently faster than the blind-search planner TLPLAN, often by orders of magnitude, and often solving problems TLPLAN is unable to solve.

There are several pieces of related research that are notable. We group this research into two overlapping categories: 1) work that compiles TEGs into classical planning problems such as that of Rintanen [20], Cresswell and Coddington [6], and recent work by the authors [4]; and 2) work that exploits automata representations of TEGs in order to plan with TEGs, such as Kabanza and Thiébaux's work on TLPLAN [14], work by Pistore and colleagues (e.g. [18, 16]), and by Edelkamp [8]. We discuss this work in the final section of this paper.

While the notion of compiling TEGs or TDCK into automata is not new, our work presents a number of significant contributions. It is the first to exploit the use of heuristic search for planning with first-order TEGs. Heuristic search results in drastic speedup in TEG planning relative to existing techniques. It is also unique in proposing a *finite* temporal logic with *FO quantifiers* for automata, together with an algorithm for their generation. This is a significant difference from previous approaches enabling us to represent goals that cannot be represented using (infinite) LTL. Moreover, if a goal is intrinsically cyclic, our translation is able to recognize (sometimes) that the goal is not achievable, which is not true of previous approaches.

2 From f-FOLTL to Parameterized NFA

In this section we present an algorithm that translates TEGs expressed as f-FOLTL sentences into parameterized, state-labeled, nondeterministic, finite automata (PSLNFA). Next we show how to simplify these automata into parameterized NFA (PNFA). We have proven the correctness of our algorithm.

2.1 f-FOLTL: Finite LTL with FO Quantifiers

We introduce f-FOLTL logic, a variant of LTL [19] which we define over *finite* rather than infinite sequences of states, and which can include first-order quantification. We use f-FOLTL formulae to describe TEGs for finite plans. f-FOLTL formulae augment LTL formulae with first-order quantification and by the use of the constant final, which is only true in final states of computation. As usual, we assume our f-FOLTL formulae are built using standard temporal and boolean connectives from a set S of symbols for predicates, functions and constants. We denote by $\mathcal{L}(S)$ the set of first-order formulae over the set of symbols S.

[2]`http://www.cs.toronto.edu/~jabaier/planningTEG/`

Definition 1 (f-FOLTL formula) *An f-*FOLTL *formula over a first-order set of symbols S is one of the following.*

1. *The 0-arity predicates* final, true *or* false,
2. $\varphi \in \mathcal{L}(S)$*, i.e. any first-order formula over S.*
3. $\neg\psi$, $\psi \wedge \chi$, $\bigcirc\psi$, *or* $\psi \mathsf{U} \chi$, *if* ψ *and* χ *are f-*FOLTL *formulae.*
4. $(\forall x)\,\varphi$, $(\exists x)\,\varphi$ *if* φ *is an f-*FOLTL *formula.*

As usual, a f-FOLTL *sentence* is a formula with no free variables. f-FOLTL formulae are interpreted over finite sequences of states, where each state is a first-order interpretation. A (finite) sequence of states $\sigma = s_0 s_1 \cdots s_n$ over a language $\mathcal{S}$ is such that s_i is a first-order interpretation, for each $i \in \{0,\ldots,n\}$ for symbols in $\mathcal{S}$ over some domain $\mathcal{D}$. Moreover each s_i assigns the same denotation to all terms of the language. As a consequence, all constants of the language refer to the same object in all interpretations. For notational convenience, we denote the suffix $s_i s_{i+1} \cdots s_n$ of σ by σ_i.

Let φ be an f-FOLTL formula. We say that $\sigma \models \varphi$ (i.e., σ is a model of φ) iff $\langle \sigma_0, V\rangle \models \varphi$, for all V, where V is an assignment of variables to domain objects. Furthermore,

- $\langle \sigma_i, V\rangle \models \mathsf{final}$ iff $i = n$.
- $\langle \sigma_i, V\rangle \models \mathsf{true}$ and $\langle \sigma_i, V\rangle \not\models \mathsf{false}$.
- $\langle \sigma_i, V\rangle \models \varphi$, where $\varphi \in \mathcal{L}(S)$ iff $\langle s_i, V\rangle \models \varphi$.
- $\langle \sigma_i, V\rangle \models \neg\varphi$ iff $\langle \sigma_i, V\rangle \not\models \varphi$.
- $\langle \sigma_i, V\rangle \models \psi \wedge \chi$ iff $\langle \sigma_i, V\rangle \models \psi$ and $\langle \sigma_i, V\rangle \models \chi$.
- $\langle \sigma_i, V\rangle \models \bigcirc\varphi$ iff $i < n$ and $\langle \sigma_{i+1}, V\rangle \models \varphi$.
- $\langle \sigma_i, V\rangle \models \psi \mathsf{U} \chi$ iff there exists a $j \in \{i,\ldots,n\}$ such that $\langle \sigma_j, V\rangle \models \chi$ and for every $k \in \{i,\ldots,j-1\}$, $\langle \sigma_k, V\rangle \models \psi$.
- $\langle \sigma_i, V\rangle \models (\forall x)\,\varphi$, iff for every $a \in \mathcal{D}$, $\langle \sigma_i, V(x/a)\rangle \models \varphi$.
- $\langle \sigma_i, V\rangle \models (\exists x)\,\varphi$, iff for some $a \in \mathcal{D}$, $\langle \sigma_i, V(x/a)\rangle \models \varphi$.

Standard temporal operators such as *always* ($\Box$), *eventually* ($\Diamond$), and *release* (R), and additional binary connectives are defined in terms of these basic elements. Recall that $\psi \mathsf{R} \chi \stackrel{\text{def}}{=} \neg(\neg\psi \mathsf{U} \neg\chi)$, and note that $\Box\varphi \stackrel{\text{def}}{=} \mathsf{false}\,\mathsf{R}\,\varphi$, and $\Diamond\varphi \stackrel{\text{def}}{=} \mathsf{true}\,\mathsf{U}\,\varphi$.

As in LTL, we can rewrite formulae containing U and R in terms of what has to hold true in the "current" state and what has to hold true in the "next" state. This is accomplished by identities 1 and 5 in the following proposition.

Proposition 1 *Suppose* φ *and* χ *are f-*FOLTL *formulae, and suppose that variable x is not free in* χ*. The following formulae are valid.*

(1) $\psi \mathsf{U} \chi \equiv \chi \vee \psi \wedge \bigcirc(\psi \mathsf{U} \chi)$, *(3)* $(\forall x)\,\varphi \mathsf{U} \chi \equiv (\forall x)\,(\varphi \mathsf{U} \chi)$,
(2) $\neg\bigcirc\varphi \equiv \mathsf{final} \vee \bigcirc\neg\varphi$, *(4)* $\chi \mathsf{U} (\exists x)\,\varphi \equiv (\exists x)\,(\chi \mathsf{U} \varphi)$,
(5) $\psi \mathsf{R} \chi \equiv \chi \wedge (\mathsf{final} \vee \psi \vee \bigcirc(\psi \mathsf{R} \chi))$.

Limiting f-FOLTL to finite computations results in several obvious discrepancies in the interpretation of LTL and f-FOLTL formulae. In particular, discrepancies can arise with LTL formulae that force their models to be infinite. For example, in f-FOLTL the formula $\Box(\varphi \supset \bigcirc\psi) \wedge \Box(\psi \supset \bigcirc\varphi)$ is equivalent to $\Box\neg(\varphi \vee \psi)$. This is because if φ or ψ were true in some state of a model, the model would be forced to be an infinite sequence of states. The reader familiar with LTL, will note that identity 2 replaces LTL's equivalence $\neg\bigcirc\varphi \equiv \bigcirc\neg\varphi$. This formula does not hold in f-FOLTL because $\bigcirc\varphi$ is true in a state iff there exists a next state that satisfies φ. Since our logic refers to finite sequences of states, the last state of each model has no successor, and therefore in such states $\neg\bigcirc\varphi$ holds for every φ.

Although there are differences between LTL and f-FOLTL, their expressive power is similar when it comes to describing temporally extended goals for finite planning. Indeed, f-FOLTL has the advantage that it is tailored to refer to finite plans. As a consequence, we can express goals that cannot be expressed with LTL. Some examples follow.

- $\Diamond(\mathsf{final} \wedge (\exists c)\,(corridor(c) \wedge at(Robot,c)))$: In the final state, $at(Robot,c)$ for some corridor c. This is one way of encoding final-state goals in f-FOLTL.
- $(\forall r_1,r_2).\,priorityOver(r_1,r_2) \supset \neg delivered(r_2)\,\mathsf{U}\, delivered(r_1) \wedge \Diamond delivered(r_2)$: If r_1 has priority over r_2 then r_2 must be delivered, but not before r_1.
- $\Diamond(p(a) \wedge \bigcirc\bigcirc\mathsf{final})$: $p(a)$ must hold true two states before the plan ends. This is an example of a goal that cannot be expressed in LTL, since it does not have the final constant.

In this paper, we present an algorithm that generates an automaton that accepts models of f-FOLTL formula expressed in *extended prenex normal form.*

Definition 2 (Extended Prenex Normal Form (EPNF))
A formula is in extended prenex normal form *(EPNF) if it is of the form $(Q_1x_1)(Q_2x_2)\cdots(Q_nx_n)\,\varphi$, where $Q_i \in \{\forall,\exists\}$ and all quantifiers that occur in φ quantify on first-order, atemporal, subformulae.*

Some formulae that are not in EPNF, have an EPNF equivalent. For example, $(\forall x)\,\Box(P(x) \supset (\exists y)\,\Diamond Q(x,y))$ is equivalent to $(\forall x)\,(P(x) \supset \Diamond(\exists y)\,Q(x,y))$, which is in EPNF. However, there are formulae that do not have an EPNF equivalent, e.g. $\Box\exists x(P(x) \wedge \Diamond Q(x))$.

2.2 The Translation to Parameterized NFA

We now present an algorithm that translates EPNF f-FOLTL sentences into parameterized, state-labeled, nondeterministic, finite automata (PSLNFA). For every LTL formula φ, there exists a Büchi automaton[3] A_φ that accepts an infinite state sequence σ if and only if $\sigma \models \varphi$ [22]. To our knowledge there is no pragmatic algorithm for translating a finite version of f-FOLTL in EPNF such as the one we use here. In this section, we propose such an algorithm and establish its correctness. Our algorithm is a modification of the one proposed in by Gerth et al. [12], and in the first stage generates a PSLNFA.

Definition 3 (PSLNFA)
A parameterized state-labeled NFA (PSLNFA) is a tuple $A = \langle Q, \Sigma(S,\mathcal{D}), \delta, L, \Gamma, \mathbf{x}, Q_0, F\rangle$, where Q is a finite set of states, and $Q_0 \subseteq Q$ is a set of initial states. The alphabet $\Sigma(S,\mathcal{D})$ is a set of first-order interpretations over the same domain $\mathcal{D}$, such that they assign the same denotation to all constant and function symbols in S, $\delta \subseteq Q \times Q$ is a transition relation, $F \subseteq Q$ is the set of final states, $\mathbf{x}$ is a string of variables, $\Gamma \in \{\forall,\exists\}^*$ is a string of quantifiers ($|\Gamma| = |\mathbf{x}|$), and the labeling function $L : Q \to 2^{\mathcal{L}(S)}$ is such that if $\varphi \in L(q)$ then all the free variables of φ are in $\mathbf{x}$.

Intuitively, a PSLNFA is like an NFA where states are labeled by sets of first-order formulae. The PSLNFA accepts a string of interpretations $s_0 \ldots s_n$ iff there is a path $q_0 \ldots q_n$ from an initial state to a final state such that labels of the states traversed are true in the corresponding interpretation (i.e., all formulas in $L(q_i)$ are true in s_i). The free variables in the labels are interpreted based on the quantifiers in Γ. Below we show that PSLNFAs accept models of f-FOLTL formulae.

Formally, a run of a PSLNFA A over a string of interpretations $\sigma = s_0 s_1 \ldots s_n$ is a finite sequence $\rho(\mathbf{a}) = q_0 q_1 \ldots q_n$, such that $q_0 \in Q_0$, $(q_i, q_{i+1}) \in \delta$ for all $i \in \{0,\ldots,n-1\}$, and $\mathbf{a}$ is a vector of objects in the domain $\mathcal{D}$. Further, for every $i \in \{0,\ldots,n\}$, $\langle s_i, \mathbf{x}/\mathbf{a}\rangle \models L(q)$. A run is *accepting* when $q_n \in F$. Let $\Gamma = V_1V_2\ldots V_n$ and $x = x_1x_2\ldots x_n$, then A accepts σ iff 'for all/for some' $a_1 \in \mathcal{D}$ and 'for all/for some' $a_2 \in \mathcal{D}$ ... and 'for all/for some' $a_n \in \mathcal{D}$ there is an accepting run $\rho(a_1a_2\ldots a_n)$. The 'for all/for some' preceding a_i is replaced by 'for all' when $V_i = \forall$, and by 'for some' otherwise.

Example PSLNFA $A = \langle\{q_0,q_1\},\Sigma,\delta,L,\Gamma,xy,\{q_0\},\{q_1\}\rangle$, where $\delta = \{(q_0,q_1),(q_1,q_1)\}$, and $L(q_0) = P(x)$ and $L(q_1) = Q(x,y)$. A accepts the models of $(\forall x,y).P(x) \wedge \bigcirc\Box Q(x,y)$ if $\Gamma = \forall\forall$ and accepts the models of $(\forall x).\,(\exists y)\,.P(x) \wedge \bigcirc\Box Q(x,y)$ in case $\Gamma = \forall\exists$. Figure 1 shows a graphical representation of a PSLNFA that can accept the models of either $(\forall x)\,\Diamond P(x)$ or $(\exists x)\,\Diamond P(x)$.

The algorithm The translation algorithm is a modification of the one proposed by Gerth et al. in [12]. In contrast to [12] it generates a PSLNFA, instead of a Büchi automaton.

To represent a node of the automaton, the algorithm uses Gerth et al.'s data structure *Node*, which is a tuple $\langle Name, Incoming, New, Old, Next\rangle$. The field *Name* : contains the name of the node; *Incoming* is the list of node names with an outgoing edge leading to *Node*; *New* contains first-order formulae that must hold at the current state but that have not been processed by the algorithm; *Old* contains the formulae that must hold in the nodes that have been processed by the algorithm; *Next* contains temporal formulae that have to be true in the immediate successors of *Node*.

In the following, suppose we want to build a PSLNFA for sentence φ in EPNF. We denote the string of quantifiers and variables at the beginning of φ by $QPrefix(\varphi)$. To generate the PSLNFA, we strip $QPrefix(\varphi)$ from φ and then leave the formula just in terms of the temporal operators U and R, and the binary boolean operators $\wedge$ and $\vee$. We then push all $\neg$'s inside such that they occur only in front of first-order formulae. The resulting formula, say, φ' is the input for the procedure we describe below. Note that the construction will start with a single node that contains φ' in its *New* field.

When processing node N, the algorithm checks whether

[3] A Büchi automaton is an extension of a finite state automaton to infinite inputs.

there are pending formulae in *New*. If there are none, then the node can be added to the *NodeSet*. Two cases can hold:

1. If there is already a node in *NodeSet* with the same fields *Old* and *Next*, then its *Incoming* list is updated by adding those nodes in N's incoming list. (Line 5).
2. If there is no such node, then N is added to *NodeSet*. Then, a new node is created for processing if $\mathsf{final} \notin Old$. This node contains N in its incoming list, and the field *New* set to N's *Next* field. The fields *Next* and *Old* of the new node are empty. (Lines 7–17).

Otherwise, if *New* is not empty, formula η is removed from *New* and added to *Old*. Then,

1. In case η is a literal, or of the form $(\forall x)\,\phi(x)$, or $(\exists x)\,\phi(x)$, then if $\neg\eta$ is in *Old*, the node is discarded (a contradiction has occurred). Otherwise, η is added to *Old* and the node continues to be processed.
2. Otherwise:
   (a) If $\eta = \varphi \wedge \psi$, both φ, and ψ are added to *New*.
   (b) If $\eta = \bigcirc\psi$, then ψ is added to *Next*.
   (c) If η is one of $\varphi \vee \psi$, $\varphi \mathsf{U} \psi$, or $\varphi \mathsf{R} \psi$, then N is split into two nodes. The set `New1`(η) and `New2`(η) are added, respectively, to the *New* field of the first and second nodes. These functions are defined as follows:

η	`New1`(η)	`New2`(η)
$\varphi \vee \psi$	$\{\varphi\}$	$\{\psi\}$
$\varphi \mathsf{U} \psi$	$\{\varphi, \bigcirc(\varphi \mathsf{U} \psi)\}$	$\{\psi\}$
$\varphi \mathsf{R} \psi$	$\{\psi, \mathsf{final} \vee \bigcirc(\varphi \mathsf{R} \psi)\}$	$\{\varphi, \psi\}$

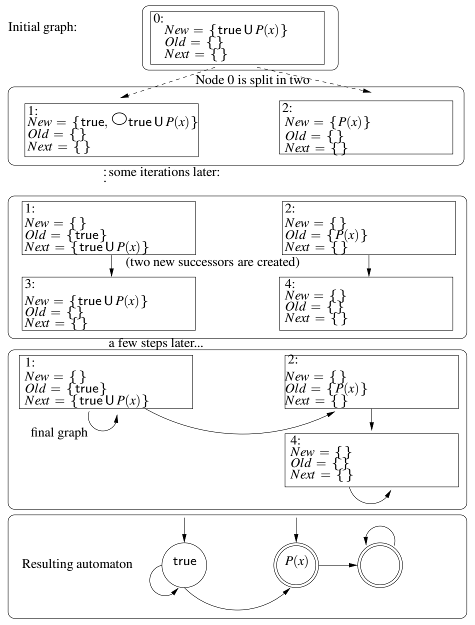
   The intuition of the split lies in standard f-FOLTL equivalences. For example, $\varphi \mathsf{U} \psi$ is equivalent to $\psi \vee (\varphi \wedge \bigcirc(\varphi \mathsf{U} \psi))$, thus one node verifies the condition ψ, whereas the other verifies $\varphi \wedge \bigcirc(\varphi \mathsf{U} \psi)$.

We define PSLNFA $A_\varphi = \langle Q, \Sigma(S,\mathcal{D}), \delta, L, \Gamma, \mathbf{x}, Q_0, F\rangle$ as follows. Let $Q = \{n \mid n \in NodeSet\}$, and $Q_0 = \{q \in Q \mid Init \in Incoming(q)\}$. Let $\mathbf{x}$ be the variables, and Γ the quantifiers in $QPrefix(\varphi)$ (preserving the order), and let $\delta(q,q')$ iff q and q' are connected in the graph. The set of final states, $F = \{q \in Q \mid Next(q) = \emptyset \text{ and } \neg\mathsf{final} \notin Old(q)\}$. Finally, let $L(q)$ be the maximal subset of $Old(q)$ that contains only literals (excluding final and $\neg\mathsf{final}$) or formulae of the form $(Qx)\,\varphi(x)$. Figure 1 shows an example of the generation of a PSLNFA for $(\forall x)\,\Diamond A(x)$.

This theorem states the correctness of the algorithm.

Theorem 1 *Let A_φ be the automaton constructed by our algorithm from an f-*FOLTL *formula φ in EPNF. Then A_φ accepts exactly the models of φ.*

An immediate consequence of this theorem is that our algorithm generates non-accepting automata for temporal formulae that are only satisfied by infinite models. Sometimes this would be reflected by the fact that the automaton does not have accepting states at all (this happens for $\varphi \wedge \Box(\varphi \supset \bigcirc\psi) \wedge \Box(\psi \supset \bigcirc\varphi)$, or by the fact that labels are inconsistent formulae (this is the case of $\Diamond P(a) \wedge (\forall x)\Box(P(x) \supset \bigcirc P(x))$). In the former case we are able to recognize that the goal is intrinsically unachievable by just looking at the



Initial graph:
0: New = {true U P(x)} Old = {} Next = {}
Node 0 is split in two
1: New = {true, ○true U P(x)} Old = {} Next = {}
2: New = {P(x)} Old = {} Next = {}
some iterations later:
1: New = {} Old = {true} Next = {true U P(x)}
2: New = {} Old = {P(x)} Next = {}
(two new successors are created)
3: New = {true U P(x)} Old = {} Next = {}
4: New = {} Old = {} Next = {}
a few steps later...
1: New = {} Old = {true} Next = {true U P(x)}
2: New = {} Old = {P(x)} Next = {}
final graph
4: New = {} Old = {} Next = {}
Resulting automaton
true
P(x)

Figure 1: Algorithm execution for formula $(\forall x)\,\Diamond P(x)$.

automata, whereas in the latter we cannot do it in general, since checking whether the labeling formulae are consistent is undecidable.

Simplifying PSLNFAs into PNFAs The algorithm presented above often produces automata that are much bigger than the optimal. To simplify the automata, we have used a modification of the algorithm presented in [9]. This algorithm uses a simulation technique to simplify the automaton. In experiments in [10], it was shown to be slightly better than LTL2AUT [7] at simplifying Büchi automata.

To apply the algorithm, we generate a parameterized NFA equivalent to the PSLNFA. Intuitively, a parameterized nondeterministic finite state automaton (PNFA) is like a PSLNFA but such that *transitions* are labeled with first-order formulae. Formally, a PNFA is a tuple $A = \langle Q, \Sigma(S,\mathcal{D}), \delta, \Gamma, \mathbf{x}, Q_0, F\rangle$, where Q, Q_0, F, Γ, $\mathbf{x}$, and $\Sigma(S,\mathcal{D})$ are defined as in PSLNFAs. A run of A over sequence of states $\sigma = s_0 s_1 \cdots s_n \in \Sigma(S,\mathcal{D})^*$ and a vector of objects $\mathbf{a}$ is $\rho(\mathbf{a}) = q_0 q_1 \cdots q_n$, where $q_0 \in Q_0$, and for some label L such that $(q_i, L, q_{i+1}) \in \delta$, $\langle s_{i+1}, \mathbf{x}/\mathbf{a}\rangle \models L$, for all $i \in \{0, \ldots, n-1\}$. Run ρ is *accepting* if $q_n \in F$. Finally, the acceptance condition is the same as in PSLNFAs.

It is straightforward to convert a PSLNFA to an equivalent PNFA by adding one initial state and copying labels of states to any incoming transition. Figure 2 shows examples of PNFAs generated by our implementation for some f-FOLTL formulae. The automaton for formula (b) is parameterized on variable x, which is indicated beside the state name.

Size complexity of the NFA In theory, the resulting automaton can be exponential in the size of formula in the worst case. Simplifications reduce the number of states of the PNFA significantly. It is critical to note that in practice,

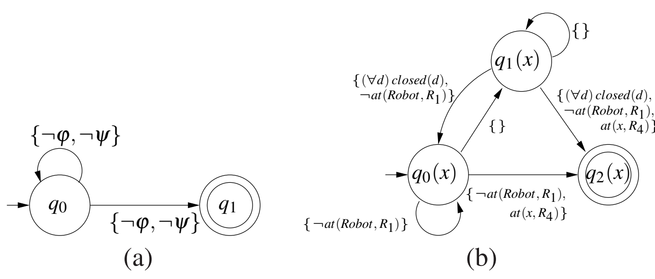
```
function Expand($Node, NodeSet$)
begin
    if $New(Node) = \emptyset$ then
        if $\exists N \in NodeSet$ and $Old(N) = Old(Node)$ and
        $Next(N) = Next(Node)$ then
            $Incoming(N) \leftarrow Incoming(N) \cup Incoming(Node)$
            return $NodeSet$
        else
            if final $\notin Old(Node)$ then
                return Expand([$Name \leftarrow$ NewName(),
                        $Incoming \leftarrow Name(Node)$,
                        $New \leftarrow Next(Node), Old \leftarrow \emptyset$
                        $Next \leftarrow \emptyset], \{Node\} \cup NodeSet$)
            else
                if $Next(Node) = \emptyset$ then
                    return $\{Node\} \cup NodeSet$
                else
                    return $NodeSet$ /* discarded */
    else
        choose $\eta \in New(Node)$
        $New(Node) \leftarrow New(Node) \setminus \{\eta\}$
        if $\eta \neq True$ and $\eta \neq False$ then
            $Old(Node) \leftarrow Old(Node) \cup \{\eta\}$
        switch $\eta$ do
            case $\eta$ is a literal, $(Qx)\,\varphi(x)$, $True$ or $False$
                if $\eta = False$ or $\neg\eta \in Old(Node)$ then
                    return $NodeSet$ /* discarded */
                else
                    return Expand($Node, NodeSet$)
            case $\eta = \bigcirc\varphi$
                $Next(Node) \leftarrow Next(Node) \cup \{\varphi\}$
                return Expand($Node, NodeSet$)
            case $\eta = \varphi \wedge \psi$
                $New(Node) \leftarrow$
                $New(Node) \cup (\{\varphi, \psi\} \setminus Old(Node))$
                return Expand($Node, NodeSet$)
            case $\eta = \varphi \vee \psi$ or $\varphi \mathsf{R} \psi$ or $\varphi \mathsf{U} \psi$
                $Node1 \leftarrow$ SplitNode($Node$, New1($\eta$))
                $Node2 \leftarrow$ SplitNode($Node$, New2($\eta$))
                return
                Expand($Node2$, Expand($Node1, NodeSet$))
end
function SplitNode ($Node, \phi$) begin
    $NewNode \leftarrow [Name \leftarrow$ NewName(),
        $Incoming \leftarrow Incoming(Node), New \leftarrow New(Node) \cup \phi$,
        $Old \leftarrow Old(Node), Next \leftarrow Next(Node)]$
    return $NewNode$
end
function GenGraph($\varphi$) begin
    Expand([$Name \leftarrow Father \leftarrow$ NewName(),
        $Incoming \leftarrow \{Init\}, New \leftarrow \{\varphi\}, Old \leftarrow \emptyset], \emptyset$)
end
```

Algorithm 1: The Algorithm

the number of states of NFAs for natural goals were generally equivalent to the size of our formulae (see Section 4).

Proposition 2 *Let φ be in negated normal form, then the number of states of A_φ is $2^{O(|\varphi|)}$.*

There are simple cases where the translation blows up; e.g., for the formula $\Diamond\phi_1 \wedge \Diamond\phi_2 \wedge \ldots \wedge \Diamond\phi_n$, the resulting NFA



$\{\neg\varphi, \neg\psi\}$
q_0
$\{\neg\varphi, \neg\psi\}$
q_1
(a)
$\{\}$
$q_1(x)$
$\{(\forall d)\, closed(d), \neg at(Robot, R_1)\}$
$\{(\forall d)\, closed(d), \neg at(Robot, R_1), at(x, R_4)\}$
$\{\}$
$q_0(x)$
$q_2(x)$
$\{\neg at(Robot, R_1), at(x, R_4)\}$
$\{\neg at(Robot, R_1)\}$
(b)

Figure 2: Simplified PNFA (a) $\Box(\varphi \supset \bigcirc\psi) \wedge \Box(\psi \supset \bigcirc\varphi)$, and (b) $\Box(at(Robot, R_1) \supset \bigcirc\Diamond(\forall d)\, closed(d)) \wedge (\forall x)\, \Diamond\Box at(x, R_4)$.

has 2^n states. Intuitively each state keeps track of a particular combination of propositions that has been true in the past.

3 Compiling PNFAs into a Planning Domain

With our TEGs translated to PNFAs, we show how PNFAs can be encoded in classical planning problems.

3.1 Planning Problems

A planning problem is a tuple $\langle \mathcal{I}, \mathcal{D}, \mathcal{G}, \mathcal{T} \rangle$, where $\mathcal{I}$ is the *initial state*, represented as a set of first-order (ground) positive facts; $\mathcal{D}$ is the *domain description*; $\mathcal{G}$ is a temporal formula describing the *goal*, and $\mathcal{T}$ is a (possibly empty) set of *derived predicate* definitions, which are predicates that are defined in terms of other fluents of the domain.

A domain description is a tuple $\mathcal{D} = \langle \mathcal{C}, \mathcal{R} \rangle$, where $\mathcal{C}$ is a set of *causal rules*, and $\mathcal{R}$ a set of *action precondition rules*. Causal rules correspond to positive and negative *effect axioms* in the situation calculus [4]. We represent positive and negative causal rules by the triple $\langle a(\mathbf{x}), c(\mathbf{x}), f(\mathbf{x}) \rangle$ and $\langle a(\mathbf{x}), c(\mathbf{x}), \neg f(\mathbf{x}) \rangle$ respectively, where $a(\mathbf{x})$ is an *action term*, $f(\mathbf{x})$ is a *fluent term*, and $c(\mathbf{x})$ is a first-order formula, each of them with free variables among those in $\mathbf{x}$. $\langle a(\mathbf{x}), \Phi(\mathbf{x}), \ell(\mathbf{x}) \rangle \in \mathcal{C}$ expresses that fluent literal $\ell(\mathbf{x})$ becomes true after performing action $a(\mathbf{x})$ in the current state if condition $\Phi(\mathbf{x})$ holds. As with ADL operators[17], the condition $c(\mathbf{x})$, can contain quantified FO subformulae. Free variables in $\mathcal{C}$ are assumed to be universally quantified.

3.2 The Compilation into a Planning Domain

We are now ready to show how the PNFA can be encoded in a planning problem. During the execution a plan $a_1 a_2 \cdots a_n$, a set of planning states $\sigma = s_0 s_1 \ldots s_n$ is generated. In what follows we make no distinction between a planning state (which are sets of ground first-order facts) and a first-order interpretation.

In the planning domain, each state of the automaton is represented by a fluent. More formally, for each state q of the automaton we add to the domain a new fluent $E_q(\mathbf{x})$. The translation is such that if a sequence of actions $a_1 a_2 \cdots a_n$ is performed in state s_0, generating the succession of states $\sigma = s_0 s_1 \ldots s_n$, then $E_q(\mathbf{c})$ is true in s_n, for a set of constants

[4]We use causal rules for simplicity. Effect axioms and ADL operators are equivalent [17].

c, if and only if there is a run $\rho(\mathbf{a})$ of A_φ on σ that ends in state q (here we implicitly assume that **c** interpret objects **a**).

Once the PNFA is modeled inside the domain, the temporal goal in the newly generated domain is reduced to a property of the final state alone. Intuitively, this property corresponds to the accepting condition of the automaton.

To represent the dynamics of the states of the automaton, there are two alternatives. The first is to modify the domain's *causal rules* to give an account of their change. The second, is to define them as *derived predicates* or *axioms*. In [4] we provided a causal rule encoding for propositional TEGs. There is an analogous encoding for the case of f-FOLTL TEGs. Henceforth, we assume the following:

- We start with a planning problem $\langle \mathcal{I}, \mathcal{D}, \mathcal{G}, \mathcal{T} \rangle$, where $\mathcal{G}$ is a temporal formula in f-FOLTL.
- Temporal goal $\mathcal{G}$ is translated to the PNFA $A_\mathcal{G} = (Q, \Sigma, \delta, \Gamma, \mathbf{x}, Q_0, F)$, with $\Gamma = V_1 \dots V_n$ and $\mathbf{x} = x_1 \dots x_n$.
- To simplify notation, we denote by $\mathtt{pred}(q)$ the set of predecessors of q. E.g., in Fig. 2(b), $\mathtt{pred}(q_0) = \{q_0, q_1\}$.
- We denote by $\lambda_{p,q}$ the formula $\bigvee_{(q,L,p)\in\delta} \bigwedge L$. E.g., in Fig. 2(b), $\lambda_{q_1,q_0} = (\forall d)\, closed(d) \wedge \neg at(Robot, R_1)$.

3.3 Translation to derived predicates (axioms)

To understand the intuition behind the translation, consider the PNFA shown in Figure 2(b). Suppose $E_{q_2}(c)$ is false in a state s_i. After performing action a_i, fluent $E_{q_2}(c)$ must become true in the resulting state, s_{i+1}, iff either $E_{q_0}(c)$ was true in s_i and $\neg at(Robot, R_1) \wedge at(c, R_4)$ is true in s_{i+1} or $E_{q_1}(c)$ was true in s_i and $\neg at(Robot, R_1) \wedge (\forall d)\, closed(D_1) \wedge at(c, R_4)$ is true in s_{i+1}. Thus, the truth value of $E_{q_2}(c)$ depends on properties that need to be verified in s_{i+1} and s_i.

In the translation we propose we write a derived predicate definition for $E_q(\mathbf{x})$. However, as we saw previously, the truth value of $E_q(\mathbf{x})$ in s_{i+1} depends on whether some fluents $E_p(\mathbf{x})$ hold true in the previous state, where p is a state of the automaton. Therefore, we need a way to represent in state s_{i+1} what fluents E_p were true in the previous state.

Thus, for each state q of the automaton we use an auxiliary fluent $Prev_q(\mathbf{x})$ which is true in a plan state s iff E_q was true in the previous state. The dynamics of fluent $Prev_q(\mathbf{x})$ is described by the following causal rules, which are added to $\mathcal{C}'$:

$$\langle a, E_q(\mathbf{x}), Prev_q(\mathbf{x}) \rangle, \qquad \langle a, \neg E_q(\mathbf{x}), \neg Prev_q(\mathbf{x}) \rangle,$$

for each action a. The following definitions are also added to $\mathcal{T}'$:

$$E_q(\mathbf{x}) \overset{\text{def}}{=} \bigvee_{p \in \mathtt{pred}(q)} Prev_p(\mathbf{x}) \wedge \lambda_{p,q}(\mathbf{x}),$$

New initial state The new initial state must specify which fluents of the form $Prev_q$ are true. These are precisely those facts that correspond to the initial state of the automaton.

$$\mathcal{I}' = \{ Prev_q(\mathbf{c}) \,|\, q \in Q_0, \mathbf{c} \text{ is a vector of domain constants} \}.$$

New goal & planning problem The new goal is defined by $\mathcal{G}' = (V_1 x_1) \dots (V_n x_n) \bigvee_{p \in F} E_p$, and the new planning problem is $\langle \mathcal{I} \cup \mathcal{I}', \mathcal{C} \cup \mathcal{C}', \mathcal{R}, \mathcal{G}', \mathcal{T} \cup \mathcal{T}' \rangle$.

Size Complexity Planning with the new translated theory is theoretically as hard as planning with the original theory. The amount of additional effort required to update newly created fluents is reflected in the size of $\mathcal{T}'$.

Proposition 3 *The size of $\mathcal{T}'$ is $O(n|Q|\ell)$ where ℓ is the maximum size of a transition in $A_\mathcal{G}$, and n is the number of action terms in the domain. The size of $\mathcal{C}'$ is only $O(n|Q|)$.*

Note that $\mathcal{T}'$ is significantly smaller than the ADL-operator based encoding proposed in [4] which is $O(n|Q|2^\ell)$. In practice, we can also show that the derived predicates provide a more efficient representation (see [3] for an experimental comparison in the propositional case).

3.4 Reducing $|Q|$

The size of the resulting translation is worst-case exponential in the number of states of the automaton, $|Q|$. We also saw that in practice it is often equivalent to the size of the formula (see Section 4). We can reduce $|Q|$ by splitting the TEG into different goals. Consider the formula $\varphi = \Diamond p_1 \wedge \ldots \wedge \Diamond p_n$, which we know has an exponential NFA. φ will be satisfied if each of the conjuncts $\Diamond p_i$ is satisfied, so instead of generating a unique NFA for φ we generate a *different* NFA *for each* $\Diamond p_i$. Then we can just plan for a goal equivalent to the conjunction of the acceptance conditions of each of those automata. The new planning problem is linear in n instead of exponential. This generalizes to any combination of boolean formulae.

3.5 Search Space Pruning by Progression

As previously noted, planners for TEGs such as TLPlan are able to prune the search space by progressing temporal formulae representing the goal. A state s is pruned by progression if the progressed temporal goal in s is equivalent to false. Intuitively, this means that there is no possible sequence of actions that when executed in s would lead to the satisfaction of the goal.

Using our approach we can also prune the search space in a similar way. We illustrate the intuition in the propositional case. Suppose we have constructed an NFA for the propositional TEG $\mathcal{G}$. Since our NFAs have no non-final states that do not lead to a final state, if at some state during the plan all fluents E_q are false for every $q \in Q$, then this means that the goal will never be satisfied. We can also do this in the first-order case by considering the quantifiers of the TEG.

In the planning domain the pruning can be achieved in two ways. One way is to add $QPrefix(\varphi) \bigvee_{q \in Q} E_q(\mathbf{x})$ as a state constraint (or *safety constraint*). The other way is to add this condition to all of the action's preconditions [1, 20].

This means that we are able to add certain types of TDCK to our planning domains by simply adding the TDCK to the goal. Currently, though, our logic does not have the Goal modality present in TLPlan which enables it to tailor the control depending on the goal.

4 Implementation and Experiments

We implemented a compiler that takes a planning domain and a TEG in EPNF f-FOLTL as input and generates a clas-

Domain	*No.*	*Problems solved*		*Speedup (s)*					*Length ratio (r)*		
	Probs.	FF$_{\mathcal{X}}$	TLPLAN	$s < 2$	$2 \leq s < 10$	$10 \leq s < 100$	$100 \leq s < 1000$	$s \geq 1000$	$r = 1$	$1 \leq r < 1.3$	$r \geq 1.3$
ZenoTravel	25	21(84%)	9(36%)	0	1(11%)	2(22%)	5(56%)	1(11%)	8(89%)	1(11%)	0
Logistics	23	23(100%)	17(74%)	1(6%)	4(24%)	4(24%)	6(35%)	2(12%)	14(82%)	2(12%)	1(6%)
Robot	16	16(100%)	9(56%)	0	4(44%)	3(33%)	2(22%)	0	5(56%)	4(44%)	0%

Table 1: Performance of our approach compared to TLPLAN in 3 benchmark domains. *Speedup* and the *length ratio* are shown for instances that were solved by both planners. *Speedup* (resp. *length ratio*) is the time taken (resp. plan length obtained) by TLPLAN over that of our approach.

sical planning problem as described in Section 4. Furthermore, the program can convert the new problem into PDDL, thereby enabling its use with a wide variety of planners.

It is hard to perform an accurate experimental analysis of our approach for two reasons. First, there are no standard benchmark problems for planning with TEGs. Second, none of the planners for TEGs is heuristic, so it is not hard to contrive problems easily solvable by our approach but completely out of the reach of non-heuristic planners.

Despite this, we have compared the performance of our translation in conjunction with FF$_{\mathcal{X}}$ against TLPLAN and the planner presented in [14] (henceforth, TPBA), which uses Büchi automata to control search. The TPBA planner is not heuristic, it is implemented in Scheme, and only supports 4 (propositional) goal templates; however, it is possible to input the (Büchi) automata for goals that do not fit in these templates. We conducted experiments that showed that our approach outperforms TPBA, for goals that fit into these templates. We do not show the results here, however [4] includes these results in the propositional case.

Table 1 presents a comparison of our approach and TLPLAN in three domains. For each domain, we designed a set of reasonably natural TEGs. Both *ZenoTravel* (a travel agency domain) and *Logistics* (a package delivery domain) are benchmark domains from past IPC. To get a feeling for the types of goals we used, here is an example of a goal in the *ZenoTravel* domain: *"persons P_1 and P_2 want to meet in some city and then eventually be at C_1 and C_2."* Our third test domain, the *Robot* domain [2] describes a robot that moves between rooms and carries objects. An example of a goal in the robot domain is: *"open all the doors, then deliver objects to the rooms, and then close all doors."*

Since most of the goals were unsolvable by TLPLAN (exceeding the 1GB RAM limit), we needed to add extra TDCK to TLPLAN so that it could show more of its potential. We conclude that our approach significantly outperforms TLPLAN. This can be seen in the *speedup* metric in the table, where a significant percentage of the problems are solved over two orders of magnitude faster. Note that in some cases, the plans that are returned are slightly longer than those obtained by TLPLAN. This is usually the case with heuristic planners, where there is a tradeoff between optimality and speed. Some plans are not solved by FF$_{\mathcal{X}}$ in the ZenoTravel domain, which is due to the presence of universally quantified disjunctive goals.

The translation times for each of these problems was very low; in most cases it was less than 15% of the planning time. Furthermore, the ratio $|A_\varphi|/|\varphi|$, where A_φ is the number of states of $|A_\varphi|$, and $|\varphi|$ is the size of the TEG φ never exceeds 1.0, which illustrates that our automata translation does not blow up easily for natural TEGs.

The results shown, although good, are not surprising. We have compared our heuristic approach to the blind-search approach (plus pruning) of TLPLAN. Consequently, these results were expected. TLPLAN is particularly good when used with classical goals and a fair amount of hand-coded TDCK. Our approach has the advantage that it is able to guide the search effectively towards the satisfaction of a TEG with no need for hand-coded TDCK.

5 Summary and Related Work

In this paper we proposed a method for translating planning problems with first-order TEGs into classical planning problems. With this translation in hand, we exploited domain-independent heuristic search to determine a plan. Our compiler generates PDDL so it is equally amenable to use with any classical planner.

There are many advantages to the quantifiers in our f-FOLTL language. In addition to providing a richer language for goal specification, f-FOLTL also results in more efficient goal processing. Propositionalizing out quantifiers, as must be done in previous approaches to this problem, increases the size of the translation as a function of the size of the domain and arity of predicates in the TEG. In particular, a propositional encoding requires grounding both the initial state of the automata and their transitions, making the compilation specific to the instance of the problem.

We tested our approach on more than 60 problems over 3 standard benchmark domains, comparing our results to TLPLAN. Using our method, the FF$_{\mathcal{X}}$ planner often produced orders of magnitude speedup compared to TLPLAN, solving some planning problems TLPLAN was unable to solve. Since FF$_{\mathcal{X}}$ propositionalizes its domains, it does not fully exploit the strength of our first-order goal encoding. We are currently integrating heuristic search into TLPLAN in order to plan with TEGs and with temporal preferences such as those expressible in PDDL3.

There are several pieces of related work. Rintanen [20] proposed a translation of a subset of LTL into a set of ADL operators, which is restricted to a very limited set of TEGs. Pistore and colleagues (e.g. [16]) used automata to encode goals for planning with model checkers. Their approach uses different goal languages and is not heuristic. Cresswell and Coddington [6] briefly outline a means of compiling LTL formulae to PDDL. They translate LTL to deterministic finite state machines (FSM) using progression

[2], and then translate the FSM into an ADL-only domain. The accepting condition must be determined by simulating an infinite repetition of the last state. Further, the use of deterministic automata makes it very prone to exponential blowup with even simple goals. The authors' code was unavailable for comparison. They report that their technique is no more efficient than TLPLAN [6], so we infer that our method is superior. Kabanza and Thiébaux's work [14] is distinct because they are able to generate infinite and cyclic plans. They compile infinite propositional LTL into a Büchi automaton. Then they use the automaton to guide planning by following a path in its graph from initial to final state, backtracking as necessary. The planner is more prone to get lost and the restriction to one automaton makes it vulnerable to blowup. In a recent poster publication [4], we have presented a similar approach for propositional TEGs. Besides the expressiveness and efficiency issues related to propositionalizing TEGs, the translation presented generates only ADL operators, which are less efficient both in theory and in practice [3]. Finally, Edelkamp [8] provides a translation of PDDL3 into PDDL2.2 by encoding propositionalized LTL hard constraints and preferences into Büchi automata. The approach cannot be used directly to provide heuristic search guidance to achieve TEGs because the acceptance condition of a Büchi automata requires visiting final states an infinite number of times.

Acknowledgements

We are grateful to Christian Fritz, for his comments on an earlier draft of this paper. We also wish to thank Froduald Kabanza, Sylvie Thiébaux, Fahiem Bacchus, and the anonymous reviewers for insightful comments on this work. This research was funded by Natural Sciences and Engineering Research Council of Canada (NSERC) grant number 229717-04.

References

[1] F. Bacchus and M. Ady. Precondition control. Unpublished manuscript, 1999.

[2] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals Mathematics Artificial Intelligence*, 22(1-2):5–27, 1998.

[3] J. A. Baier and S. A. McIlraith. Planning with propositional temporally extended goals using heuristic search. Technical Report CSRG-537, Department of Computer Science, University of Toronto, Toronto, Canada, 2006.

[4] J. A. Baier and S. A. McIlraith. Planning with temporally extended goals using heuristic search. In *Proc. of the 16th Intl. Conference on Automated Planning and Scheduling (ICAPS-06)*, 2006. To appear.

[5] A. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[6] S. Cresswell and A. M. Coddington. Compilation of LTL goal formulas into PDDL. In R. L. de Mántaras and L. Saitta, editors, *Proc. of the 16th European Conference on Artificial Intelligence (ECAI-04)*, pages 985–986, Valencia, Spain, August 2004. IOS Press.

[7] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Proc. of the 11th Intl. Conference on Computer Aided Verification (CAV-99)*, volume 1633 of *LNCS*, pages 249–260, Trento, Italy, 1999. Springer.

[8] S. Edelkamp. On the compilation of plan constraints and preferences. In *Proc. of the 16th Intl. Conference on Automated Planning and Scheduling (ICAPS-06)*, 2006. To appear.

[9] K. Etessami and G. J. Holzmann. Optimizing Büchi automata. In *Proc. of the 11th Intl. Conference on Concurrency Theory (CONCUR-00)*, volume 1877 of *LNCS*, pages 153–167, University Park, PA, August 2000. Springer.

[10] C. Fritz. Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In *Proc. of the 8th Intl. Conference on Implementation and Application of Automata (CIAA-03)*, volume 2759 of *LNCS*, pages 35–48, Santa Barbara, CA, 2003. Springer.

[11] A. Gerevini and D. Long. Plan constraints and preferences for PDDL3. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy, 2005.

[12] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. of the 15th Intl. Symposium on Protocol Specification, Testing and Verification (PSTV-95)*, pages 3–18, Warsaw, Poland, July 1995.

[13] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[14] F. Kabanza and S. Thiébaux. Search control in planning for temporally extended goals. In *Proc. of the 15th Intl. Conference on Automated Planning and Scheduling (ICAPS-05)*, Monterey, CA, USA, June 2005.

[15] J. Kvarnström and P. Doherty. Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics Artificial Intelligence*, 30(1-4):119–169, 2000.

[16] U. D. Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *Proc. of AAAI/IAAI*, pages 447–454, Edmonton, Alberta, Canada, 2002.

[17] E. P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. of the 1st International Conference of Knowledge Representation and Reasoning (KR-89)*, pages 324–332, Toronto, Canada, May 1989.

[18] M. Pistore, R. Bettin, and P. Traverso. Symbolic techniques for planning with extended goals in non-deterministic domains. In *Proc. of the 6th European Conference on Planning (ECP-01)*, Toledo, Spain, 2001.

[19] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symposium on Foundations of Computer Science (FOCS-77)*, pages 46–57, 1977.

[20] J. Rintanen. Incorporation of temporal logic control into plan operators. In W. Horn, editor, *Proc. of the 14th European Conference on Artificial Intelligence (ECAI-00)*, pages 526–530, Berlin, Germany, August 2000. IOS Press.

[21] S. Thiébaux, J. Hoffmann, and B. Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168(1-2):38–69, 2005.

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