Planning with Temporally Extended Goals using Heuristic Search

Jorge A. Baier and **Sheila A. McIlraith**
Deptartment of Computer Science, University of Toronto,
Toronto, ON, M5S 3H5, Canada

Abstract

Temporally extended goals (TEGs) refer to properties that must hold over intermediate and/or final states of a plan. Current planners for TEGs prune the search space during planning via goal progression. However, the fastest classical domain-independent planners rely on heuristic search. In this paper we propose a method for planning with propositional TEGs using heuristic search. To this end, we translate an instance of a planning problem with TEGs into an equivalent classical planning problem. With this translation in hand, we exploit heuristic search to determine a plan. We represent TEGs using propositional linear temporal logic which is interpreted over finite sequences of states. Our translation is based on the construction of a nondeterministic finite automaton for the TEG. We prove the correctness of our algorithm and analyze the complexity of the resulting representation. The translator is fully implemented and available. Our approach consistently outperforms existing approaches to planning with TEGs, often by orders of magnitute.

1 Introduction

In this paper we address the problem of generating finite plans for temporally extended goals (TEGs) using heuristic search. TEGs refer to properties that must hold over intermediate and/or final states of a plan. From a practical perspective, TEGs 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 sequence (e.g., book flight after confirming hotel availability), safety goals (e.g., the door must always be open), and achieving a goal within some number of steps (e.g., at most 3 states after lifting a heavy object, the robot must recharge its batteries).

Planning with TEGs is fundamentally different from using temporally extended domain control knowledge to guide search (e.g., TLPLAN [1], and TALPLAN [11]). TEGs express properties of the plan we want to generate, whereas domain control knowledge expresses general properties of the search for a class of plans [10]. As a consequence, domain control knowledge is generally associated with an additional final-state goal.

A strategy for planning with TEGs, as exemplified by TLPLAN, is to use some sort of blind search on a search space that is constantly pruned by the progression of the temporal goal formula. This works well for safety-oriented goals (e.g., $\Box open(door)$) because it prunes those actions that falsify the goal. Nevertheless, it is less effective with respect to liveness properties such as $\Diamond at(Robot, Home)$. Our objective is to exploit heuristic search to efficiently generate plans with TEGs.

To achieve this, we convert a TEG planning problem into a classical planning problem where the goal is expressed in terms of the final state, and then we use existing heuristic search techniques. An advantage of this approach is that we can use any heuristic planner with the resulting problem.

In contrast to previous approaches, we propose to represent TEGs in f-LTL, a version of propositional linear temporal logic (LTL) [14] which can only be interpreted by finite computations, and is more natural for expressing properties of finite plans. To convert a TEG to a classical planning problem we provide a translation of f-LTL formulae to nondeterministic finite automata (NFA). We prove the correctness of our algorithm. We analyze the space complexity of our translations and suggest techniques to reduce space.

Our translator is fully implemented and available on the Web. It outputs PDDL problem descriptions, which makes our approach amenable to use with a variety of classical planners. We have experimented with the heuristic planner FF [9]. Our experimental results illustrate the significant power heuristic search brings to planning with TEGs. In almost all of our experiments, we outperform existing (non-heuristic) techniques for planning with TEGs.

There are several papers that addressed related issues. First is work that compiles TEGs into classical planning problems such as that of Rintanen [15], and Cresswell and Coddington [3]. Second is work that exploits automata representations of TEGs in order to plan with TEGs, such as Kabanza and Thiébaux's work on TLPLAN [10] and work by Pistore and colleagues [12]. We discuss this work in the final section of this paper.

2 Preliminaries

We represent TEGs using f-LTL logic, a variant of a propositional LTL [14] which we define over *finite* rather than infinite sequences of states. f-LTL formulae augment LTL formulae with the propositional constant final, which is only true in final states of computation. An f-LTL formula over a set $\mathcal{P}$ of propositions is (1) final, true, false, or p, for any $p \in \mathcal{P}$; or (2) $\neg\psi$, $\psi \wedge \chi$, $\bigcirc\psi$, or $\psi \mathsf{U} \chi$, if ψ and χ are f-LTL formulae.

The semantics of an f-LTL formula over $\mathcal{P}$ is defined over finite sequences of states $\sigma = s_0 s_1 \cdots s_n$, such that $s_i \subseteq \mathcal{P}$, for each $i \in \{0, \ldots, n\}$. We denote the suffix $s_i \cdots s_n$ of σ by σ_i. Let φ be an f-LTL formula. We say that $\sigma \models \varphi$ iff $\sigma_0 \models \varphi$. Furthermore,

- $\sigma_i \models \mathsf{final}$ iff $i = n$, $\sigma_i \models \mathsf{true}$, $\sigma_i \not\models \mathsf{false}$, and $\sigma_i \models p$ iff $p \in s_i$

- $\sigma_i \models \neg\varphi$ iff $\sigma_i \not\models \varphi$, and $\sigma_i \models \psi \wedge \chi$ iff $\sigma_i \models \psi$ and $\sigma_i \models \chi$.
- $\sigma_i \models \bigcirc\varphi$ iff $i < n$ and $\sigma_{i+1} \models \varphi$.
- $\sigma_i \models \psi \mathsf{U} \chi$ iff there exists a $j \in \{i,\ldots,n\}$ such that $\sigma_j \models \chi$ and for every $k \in \{i,\ldots,j-1\}$, $\sigma_k \models \psi$.

Standard temporal operators such as *always* ($\Box$), *eventually* ($\Diamond$), and *release* (R), and additional binary connectives such as $\vee$, $\supset$ and $\equiv$ can be defined in terms of the basic elements of the language (e.g., $\psi \mathsf{R} \chi \stackrel{\text{def}}{=} \neg(\neg\psi \mathsf{U} \neg\chi)$).

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. The following f-LTL identities are the basis for our translation algorithm.
1. $\psi \mathsf{U} \chi \equiv \chi \vee \psi \wedge \bigcirc(\psi \mathsf{U} \chi)$. 3. $\neg\bigcirc\varphi \equiv \mathsf{final} \vee \bigcirc\neg\varphi$.
2. $\psi \mathsf{R} \chi \equiv \chi \wedge (\mathsf{final} \vee \psi \vee \bigcirc(\psi \mathsf{R} \chi))$.

Identities 2 and 3 explicitly mention the constant final. Those familiar with LTL, will note that identity 3 replaces LTL's equivalence $\neg\bigcirc\varphi \equiv \bigcirc\neg\varphi$. In f-LTL $\bigcirc\varphi$ is true in a state iff there exists a next state that satisfies φ. Since our logic is finite, the last state of each model has no successor, and therefore in such states $\neg\bigcirc\varphi$ holds for every φ.

The expressive power of f-LTL is similar to that of LTL when describing TEGs. Indeed, f-LTL has the advantage that it is tailored to refer to finite plans, and therefore we can express goals that cannot be expressed with LTL. Some examples of TEGs follow.
- $\Box(\mathsf{final} \supset at(Robot,R1))$: In the final state, $at(Robot,R1)$ must hold. This is one way of encoding final-state goals.
- $\Diamond(p \wedge \bigcirc\bigcirc\mathsf{final})$: p 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.

Planning Problems A planning problem is a tuple $\langle \mathcal{I},\mathcal{D},\mathcal{G}\rangle$, where set $\mathcal{I}$ is the *initial state*, composed by first-order (ground) positive facts; $\mathcal{D}$ is the *domain description*; $\mathcal{G}$ is a TEG in f-LTL.

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*. Intuitively, a causal rule defines when a fluent literal becomes true after performing an action. We represent causal rules by the triple $\langle a(\vec{x}), c(\vec{x}), \ell(\vec{x})\rangle$, where $a(\vec{x})$ is an *action term*, $\ell(\vec{x})$ is a *fluent literal*, and $c(\vec{x})$ is a first-order formula, each of them with free variables among those in $\vec{x}$. Intuitively, $\langle a(\vec{x}), c(\vec{x}), \ell(\vec{x})\rangle$ expresses that $\ell(\vec{x})$ becomes true after performing action $a(\vec{x})$ in the current state if condition $c(\vec{x})$ holds true. As with ADL operators, the condition $c(\vec{x})$, can contain quantified first-order subformulae. Moreover, ADL operators can be constructed from causal rules and vice versa [13]. Finally, we assume that for each action term and fluent term, there exists at most one positive and one negative causal rule in $\mathcal{C}$. All free variables in rules of $\mathcal{C}$ or $\mathcal{R}$ are regarded as universally quantified.

Regression The causal rules of a domain describe the dynamics of individual fluents. However, to model an NFA in a planning domain, we need also know the dynamics of arbitrary complex formulae, such as for example, the causal rule for $at(o,R_1) \wedge holding(o)$. This is normally accomplished by goal regression [18, 13]. For example, if the following are causal rules for fluents α and β:

$$\langle a, \Phi^+_{a,\alpha}, \alpha\rangle, \quad \langle a, \Phi^-_{a,\alpha}, \neg\alpha\rangle, \quad \langle a, \Phi^+_{a,\beta}, \beta\rangle \quad \langle a, \Phi^-_{a,\beta}, \neg\beta\rangle,$$

we would add the causal rule $\langle a, \Phi^+, \alpha \wedge \beta\rangle$ for $\alpha \wedge \beta$, where $\Phi^+ = \{\Phi^+_{a,\alpha} \wedge \Phi^+_{a,\beta}\} \vee \{\alpha \wedge \neg\Phi^-_{a,\alpha} \wedge \Phi^+_{a,\beta}\} \vee \{\beta \wedge \neg\Phi^-_{a,\beta} \wedge \Phi^+_{a,\alpha}\}$
The size of the resulting causal rule (before simplification) for a boolean combination of fluents can grow exponentially:

Proposition 1 *Let $\varphi = f_0 \wedge f_1 \wedge \ldots \wedge f_n$. Then, assuming no simplifications are made, $|\Phi^+_{a,\varphi}| = \Omega(3^n(m^+ + m^-))$, where $m^+ = \min_i |\Phi^+_{a,f_i}|$, and $m^- = \min_i |\Phi^-_{a,f_i}|$.*

Moreover, the simplification of a boolean formula is also exponential in the size of the formula. Despite this bad news, below we present a technique to reduce the size of the resulting translation for formulae like these.

3 Translating f-LTL to NFA

It is well-known that for every LTL formula φ, there exists a Büchi automaton A_φ, such that it accepts an infinite state sequence σ iff $\sigma \models \varphi$ [17, 16]. To our knowledge, there exists no pragmatic algorithm for translating a finite version of LTL such as the one we use here[1]. To this end, we have designed an algorithm based on the one proposed by Gerth et al. [7]. The automaton generated is a *state-labeled NFA* (SLNFA), i.e. an NFA where states are labeled with formulae. Given a finite state sequence $\sigma = s_0 \ldots s_n$, the automaton goes through a path of states $q_0 \ldots q_n$ iff the formula label of q_i is true in s_i. The automaton accepts σ iff q_n is final.

Space precludes displaying the complete algorithm. Nevertheless, the code is downloadable from the Web[2], and the algorithm is described in detail in [2]. Briefly, there are three main modifications to the algorithm of Gerth et al [7]. First, the generation of successors now takes into account the final constant. Second, the *splitting* of the nodes is done considering f-LTL identities in Section 2 instead of standard LTL identities. Third, the acceptance condition of the automaton is defined using the constant final and the fact that the logic is interpreted over finite sequences of states. We prove that our algorithm is correct:

Theorem 1 *Let A_φ be the automaton built by the algorithm from φ. Then A_φ accepts exactly the set of computations that satisfy φ.*

Simplifying SLNFAs into NFAs Our algorithm often produces automata that are much bigger than the optimal. To simplify it, we have used a modification of the algorithm presented in [5]. This algorithm uses a simulation technique to simplify the automaton. In experiments in [6], it was shown to be slightly better than LTL2AUT [4] at simplifying Büchi automata. The resulting automaton is an NFA, as the ones shown in Figure 1. In contrast to SLNFA, in NFA transitions are labeled with formulae.

Size complexity of the NFA Although simplifications normally reduce the number of states of the NFA significantly, the resulting automaton can be exponential in the size of the formula in the worst case. E.g., for the formula $\Diamond p_1 \wedge \Diamond p_2 \wedge \ldots \wedge \Diamond p_n$, the resulting NFA has 2^n states. Below, we see that this is not a limitation in practice.

[1] In [8], finite automata are built for a $\bigcirc$-free subset of LTL, that does not include the final constant.

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

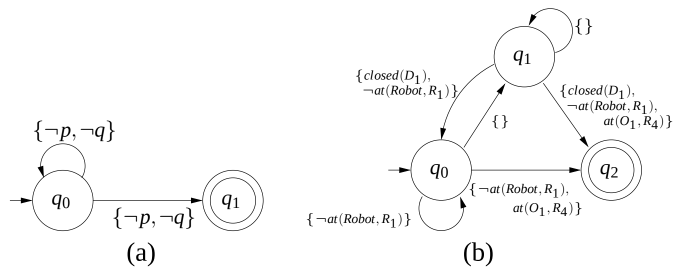
$\{\neg p, \neg q\}$
q_0
$\{\neg p, \neg q\}$
q_1
(a)
$\{\}$
q_1
$\{closed(D_1), \neg at(Robot, R_1)\}$
$\{\}$
$\{closed(D_1), \neg at(Robot, R_1), at(O_1, R_4)\}$
q_0
q_2
$\{\neg at(Robot, R_1), at(O_1, R_4)\}$
$\{\neg at(Robot, R_1)\}$
(b)

Figure 1: Simplified NFA for (a) $\Box(p \supset \bigcirc q) \wedge \Box(q \supset \bigcirc p)$, and (b) $\Box(at(Robot, R_1) \supset \bigcirc\Diamond closed(D_1)) \wedge \Diamond\Box at(O_1, R_4)$.

4 Compiling NFAs into a Planning Domain

Now that we are able to represent TEGs as NFAs, we show how the NFA can be encoded into a planning problem. Once the NFA is modeled inside the domain, the temporal goal in the newly generated domain in reduced to a property of the final state alone. Intuitively, this property corresponds to the accepting condition of the automaton.

In the rest of the section, we assume the following. We start with a planning problem $L = \langle \mathcal{I}, \mathcal{D}, \mathcal{G} \rangle$, where $\mathcal{G}$ is a TEG in f-LTL. The NFA $A_{\mathcal{G}} = (Q, \Sigma, \delta, Q_0, F)$ is built for $\mathcal{G}$. We denote by $\lambda_{p,q}$ the formula $\bigvee_{(q,L,p)\in\delta} \bigwedge L$. E.g., in Fig. 1(b), $\lambda_{q_1,q_0} = closed(D_1) \wedge \neg at(Robot, R_1)$. Finally, we denote by $Pred(q)$ the states that are predecessors of q.

In the planning domain, each state of the NFA is represented by a fluent. For each state q we add to the domain a new fluent E_q. The translation is such that if 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 is true in s_n if and only if there is a run of $A_{\mathcal{G}}$ on σ that ends in state q.

For each fluent E_q we generate a new set of causal rules. New rules are added to the set $\mathcal{C}'$, which is initialized to $\emptyset$.

For each action a, we add to $\mathcal{C}'$ the causal rules $\langle a, \Phi^+_{a,E_q}, E_q \rangle$ and $\langle a, \Phi^-_{a,E_q}, \neg E_q \rangle$ where:

$$\begin{aligned}
\Phi^+_{a,E_q} &= \bigvee_{p \in Pred(q)\setminus\{q\}} E_p \wedge (\Phi^+_{a,\lambda_{p,q}} \vee (\lambda_{p,q} \wedge \neg\Phi^-_{a,\lambda_{p,q}})),\\
\Phi^-_{a,E_q} &= \neg\Phi^+_{a,E_q} \wedge \neg(\Phi^+_{q,\lambda_{q,q}} \vee \lambda_{q,q} \wedge \neg\Phi^-_{a,\lambda_{q,q}}).
\end{aligned}$$

where $\Phi^+_{a,\lambda_{p,q}}$ (resp. $\Phi^-_{a,\lambda_{p,q}}$) is the condition under which a makes $\lambda_{p,q}$ true (resp. false). Both formulae must be obtained via regression. Formula $\lambda_{q,q}$ is false if there is no self transition in q.

The initial state must give an account of which fluents E_q are initially true. The new set of facts $\mathcal{I}'$ is the following $\mathcal{I}' = \{E_q \mid (p, L, q) \in \delta, p \in Q_0, L \subseteq \mathcal{I}\}$.

Intuitively, the automaton $A_{\mathcal{G}}$ accepts iff the temporally extended goal $\mathcal{G}$ is satisfied. Therefore, the new goal, $\mathcal{G}' = \bigvee_{p\in F} E_p$, is defined according to the acceptance condition of the NFA. The final planning problem L' is $\langle \mathcal{I} \cup \mathcal{I}', \mathcal{C} \cup \mathcal{C}', \mathcal{R}, \mathcal{G}' \rangle$.

Size complexity The size of the translated domain has a direct influence on how hard it is to plan with that domain. We can prove that the size of the translated domain is $O(n|Q|2^{\ell})$, where ℓ is the maximum size of a transition in $A_{\mathcal{G}}$, n is the number of action terms in the domain, and $|Q|$ is the number of states of the automaton.

Prb. #	Comp. time	No. Sts.	$\lvert\mathcal{G}\rvert$	FF		TLPLAN		
				t	ℓ	t	t-ctrl	ℓ
1	.02	2	6	.02	6	.07	.01	6
2	.02	2	6	.02	8	.04	.03	8
3	.09	15	21	.04	10	.20	.02	10
4	.06	5	12	.03	6	.38	.10	6
5	.07	6	21	.04	15	.5	.19	13
6	.49	37	71	.19	16	.51	.17	18
7	.05	6	21	.05	9	.96	.31	10
8	.07	15	9	.05	10	1.40	.04	10
9	.01	4	11	.03	18	13.90	.15	14
10	.04	6	12	.07	32	17.52	.40	14
11	.08	5	23	.06	22	m	m	–
12	.09	5	25	.50	25	m	m	–
13	.09	6	15	m	–	m	m	–
14	.32	5	31	m	–	m	m	–
15	.07	5	18	.11	31	m	m	–
16	.09	10	22	m	–	m	m	–

(a)

Prb.	FF		TPBA+c	
	t	ℓ	t	ℓ
1	0.02	2	0.24	2
2	0.02	5	0.96	5
3	0.01	5	1.3	5
4	0.02	7	3.29	7
5	0.02	10	11.66	10
6	0.02	12	28.87	12
7	0.02	15	82.57	15
8	0.02	19	35.69	17
9	0.02	21	13.37	20
10	0.23	52	126.25	35
11	0.07	54	m	–
12	0.23	47	m	–
13	0.54	72	m	–
14	4.03	82	m	–
15	11.19	95	m	–

(b)

Table 1: Our approach compared to TLPLAN (a) and search control with Büchi automata (b)

Reducing $|Q|$ We previously saw that $|Q|$ can be exponential in the size of the formula. Fortunately, there is a workaround. Consider for example the formula $\varphi = \Diamond p_1 \wedge \ldots \wedge \Diamond p_n$, which we know has an exponential NFA. φ is satisfied if each of the conjuncts $\Diamond p_i$ is satisfied. Instead of generating a unique NFA for φ we generate different NFA *for each* $\Diamond p_i$. Then we plan for a goal equivalent to the conjunction of the acceptance conditions of each of those automata. This generalizes to any combination of boolean formulae.

5 Implementation and Experiments

We implemented a compiler that given a domain and a f-LTL TEG, generates a classical planning problem following Section 4. The compiler can further convert the new problem into a PDDL domain/problem, thereby enabling its use with a variety of available planners.

We conducted several experiments in the Robots Domain [1] to test the effectiveness of our approach. In each experiment, we compiled the planning problem to PDDL. To evaluate the translation we used the FF planner.

Table 1(a) presents results obtained for various temporal goals by our translation and TLPLAN. The second column shows the time taken by the translation, the third shows the number of states of the automata representing the goal, and the fourth shows the size of the goal formula, $|\mathcal{G}|$. The rest of the columns show the time (t) and length (ℓ) of the plans for each approach. In the case of the TLPLAN, two times are presented. In the first (t) no additional search control was added to the planner, i.e. the planner was using only the goal to prune the search space. In the second (t-ctrl) we added (by hand) additional control information to "help" TLPLAN do a better search. The character 'm' stands for *ran out of memory*.

Our approach significantly outperformed TLPLAN. TLPLAN is only competitive in very simple cases. In most cases, our approach is one or two orders of magnitude faster than TLPLAN. Moreover, the number of automata states is comparable to the size of the goal formula, which illustrates that our approach does not blow up easily for natural TEGs. We also observe that FF cannot solve all problems. This is because FF transforms the domain to a STRIPS problem, and tends to blow up when conditional effects contain large formulae. This problem, can be overcome if one uses *derived predicates* in the translation as proposed in [2].

Table 1(b) compares our approach's performance to that of the planner presented in [10] (henceforth, TPBA), which uses Büchi automata to control search. In this case we used goals of the form $\Diamond(p_1 \wedge \bigcirc(\Diamond p_2 \wedge \ldots \wedge \bigcirc\Diamond p_n)\ldots)$, which is one of the four goal templates supported by this planner. Again, our approach significantly outperforms TPBA, even in the presence of extra control information added by hand (this is indicated by the '+c' in the table).

The results presented above are not surprising. None of the planners we have compared to uses heuristic search, which means they may not have enough information to determine which action to choose during search. The TLPLAN family of planners is particularly efficient when control information is added to the planner. Usually this information is added by an expert in the planning domain. However, control information, while natural for classical goals, may be hard to write for temporally extended goals. The advantage of our approach is that we do not need to write this information to be efficient. Moreover, control information can also be added in the context of our approach by integrating it into the goal formula.

6 Discussion and Related Work

In this paper we proposed a method to generate plans for TEGs using heuristic search. We proposed a translation method that takes as input a planning problem with an f-LTL TEG and produces a classical planning problem. Experimental results demonstrate that our approach outperforms—often by several orders of magnitude—existing (non-heuristic) planners for TEGs in the Robots Domain. [2]. Our approach is limited to propositional TEGs. In [2] we show how we can extend it to capture a compelling subset of first-order f-LTL. We also provide analogous performance results on multiple domains.

There are several notable pieces of related work. TPBA, the temporal extension of TLPLAN that uses search control, and that we use in our experiments [10], constructs a Büchi automaton to represent the goal. It then uses the automaton to guide planning by following a path in its graph from an initial to final state, setting transition labels as subgoals, and backtracking as necessary.

Approaches for planning as symbolic model checking have also used automata to encode the goals (e.g. [12]). These approaches use different languages for extended goals, and are not heuristic.

In [3] a translation of LTL formulae to PDDL has been proposed. They translate LTL formulae to a *deterministic* finite state machine (FSM). The FSM is generated by successive applications of the *progress* operator of [1] to the TEG. The use of deterministic automata makes it prone to exponential blowup even with simple goals, e.g., $\Diamond(p \wedge \bigcirc^n q)$. The authors' code was unavailable for comparison with our work. Nevertheless, they report that their technique is no more efficient than TLPLAN, so we infer that our approach has better performance.

Finally, [15] proposes a translation of a subset of LTL into a set of ADL operators. Their translation does not use automata, and therefore is limited to a small subset of LTL.

Acknowledgments We are extremely grateful to Froduald Kabanza, who provided the TPBA code for comparison. We also wish to thank Sylvie Thiébaux, Fahiem Bacchus, and the anonymous reviewers for insightful comments on this work. Finally, we gratefully acknowledge funding from NSERC Research Grant 229717-04.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Math Art. Int.*, 22(1-2):5–27, 1998.

[2] J. A. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. Forthcoming.

[3] S. Cresswell and A. Coddington. Compilation of LTL goal formulas into PDDL. In *ECAI-04*, pages 985–986, 2004.

[4] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Proc. CAV-99*, volume 1633 of *LNCS*, pages 249–260, Trento, Italy, 1999.

[5] K. Etessami and G. J. Holzmann. Optimizing büchi automata. In *Proc. CONCUR-2000*, pages 153–167, 2000.

[6] C. Fritz. Constructing Büchi automata from ltl using simulation relations for alternating büchi automata. In *Proc. CIAA 2003*, volume 2759 of *LNCS*, pages 35–48, 2003.

[7] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV-95*, pages 3–18, 1995.

[8] Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *Proc. ASE-01*, pages 412–416, 2001.

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

[10] F. Kabanza and S. Thiébaux. Search control in planning for temporally extended goals. In *Proc. ICAPS-05*, 2005.

[11] J. Kvarnström and P. Doherty. Talplanner: A temporal logic based forward chaining planner. *Ann. of Math Art. Int.*, 30(1-4):119–169, 2000.

[12] U. Dal Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *Proc. AAAI/IAAI*, pages 447–454, 2002.

[13] E. P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR-89*, pages 324–332, 1989.

[14] A. Pnueli. The temporal logic of programs. In *Proc. FOCS-77*, pages 46–57, 1977.

[15] J. Rintanen. Incorporation of temporal logic control into plan operators. In *Proc. ECAI-00*, pages 526–530, 2000.

[16] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, volume 1043 of *LNCS*, pages 238–266. Springer, 1995.

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

[18] R. Waldinger. Achieving several goals simultaneously. In *Mach. Intel. 8*, pages 94–136. Ellis Horwood, 1977.