Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals

Jorge Torres and **Jorge A. Baier**
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract

Linear temporal logic (LTL) is an expressive language that allows specifying temporally extended goals and preferences. A general approach to dealing with general LTL properties in planning is by "compiling them away"; i.e., in a pre-processing phase, all LTL formulas are converted into simple, non-temporal formulas that can be evaluated in a planning state. This is accomplished by first generating a finite-state automaton for the formula, and then by introducing new fluents that are used to capture all possible runs of the automaton. Unfortunately, current translation approaches are worst-case exponential on the size of the LTL formula. In this paper, we present a polynomial approach to compiling away LTL goals. Our method relies on the exploitation of alternating automata. Since alternating automata are different from non-deterministic automata, our translation technique does not capture all possible runs in a planning state and thus is very different from previous approaches. We prove that our translation is sound and complete, and evaluate it empirically showing that it has strengths and weaknesses. Specifically, we find classes of formulas in which it seems to outperform significantly the current state of the art.

1 Introduction

Linear Temporal Logic (LTL) [Pnueli, 1977] is a compelling language for the specification of goals in AI planning, because it allows defining constraints on state trajectories which are more expressive than simple final-state goals, such as "*deliver priority packages before non-priority ones*", or "*while moving from the office to the kitchen, make sure door D becomes closed some time after it is opened*". It was first proposed as the goal specification language of TLPlan system [Bacchus and Kabanza, 1998]. Currently, a limited but compelling subset of LTL has been incorporated into PDDL3 [Gerevini et al., 2009] for specifying hard and soft goals.

While there are some systems that natively support the PDDL3 subset of LTL [e.g., Coles and Coles, 2011], when planning for general LTL goals, there are two salient approaches: goal progression [Bacchus and Kabanza, 1998] and compilation approaches [Rintanen, 2000; Cresswell and Coddington, 2004; Edelkamp, Jabbar, and Naizih, 2006; Baier and McIlraith, 2006]. Goal progression has been shown to be extremely effective when the goal formula encodes some domain-specific control knowledge that prunes large portions of the search space [Bacchus and Kabanza, 2000]. In the absence of such expert knowledge, however, compilation approaches are more effective at planning for LTL goals since they produce an equivalent classical planning problem, which can then be fed into optimized off-the-shelf planners.

State-of-the-art compilation approaches to planning for LTL goals exploit the relationship between LTL and finite-state automata (FSA) [Edelkamp, 2006; Baier and McIlraith, 2006]. As a result, the size of the output is worst-case *exponential* in the size of the LTL goal. Since deciding plan existence for both LTL and classical goals is PSPACE-complete [Bylander, 1994; De Giacomo and Vardi, 1999], none of these approaches is optimal with respect to computational complexity, since they rely on a potentially exponential compilation. From a practical perspective, this worst case is also problematic since the size of a planning instance has a direct influence on planning runtime.

In this paper, we present a novel approach to compile away general LTL goals into classical goals that runs in polynomial time on the size of the input that is thus optimal with respect to computational complexity. Like existing FSA approaches, our compilation exploits a relation between LTL and automata, but instead of FSA, we exploit alternating automata (AA), a generalization of FSA that does not seem to be efficiently compilable with techniques used in previous approaches. Specifically, our compilation handles each non deterministic choice of the AA with a specific action, hence leaving non-deterministic choices to be decided at planning time. This differs substantially from both Edelkamp's and Baier and McIlraith's approaches, which represent all runs of the automaton simultaneously in a single planning state.

We propose variants of our method that lead to performance improvements of planning systems utilizing relaxed-plan heuristics. Finally, we evaluate our compilation empirically, comparing it against Baier and McIlraith's—who below we refer to as B&M. We conclude that our translation has strengths and weaknesses: it outperforms B&M's for classes of formulas that require very large FSA, while B&M's seems stronger for shallower, simpler formulas.

In the rest of the paper, we outline the required background, we describe our AA construction for finite LTL logic, and then show the details of our compilation approach. We continue describing the details of our empirical evaluation. We finish with conclusions.

2 Preliminaries

The following sections describe the background necessary for the rest of the paper.

2.1 Propositional Logic Preliminaries

Given a set of propositions F, the set of *literals* of F, $Lit(F)$, is defined as $Lit(F) = F \cup \{\neg p \mid p \in F\}$. The complement of a literal ℓ is denoted by $\overline{\ell}$, and is defined as $\neg p$ if $\ell = p$ and as p if $\ell = \neg p$, for some $p \in F$. $\overline{L}$ denotes $\{\overline{\ell} \mid \ell \in L\}$.

Given a Boolean value function $\pi : P \to \{\mathsf{false}, \mathsf{true}\}$, and a Boolean formula φ over P, $\pi \models \varphi$ denotes that π satisfies φ, and we assume it defined in the standard way. To simplify notation, we use $s \models \varphi$, for a set s of propositions, to abbreviate $\pi_s \models \varphi$, where $\pi_s = \{p \to \mathsf{true} \mid p \in s\} \cup \{p \to \mathsf{false} \mid p \in F \setminus s\}$. In addition, we say that $s \models R$, when R is a set of Boolean formulas, iff $s \models r$, for every $r \in R$.

2.2 Deterministic Classical Planning

Deterministic classical planning attempts to model decision making of an agent in a deterministic world. We use a standard planning language that allows so-called negative preconditions and conditional effects. A *planning problem* is a tuple $\langle F, O, I, G\rangle$, where F is a set of propositions, O is a set of action operators, $I \subseteq F$ defines an initial state, and $G \subseteq Lit(F)$ defines a goal condition.

Each action operator a is associated with the pair $(prec(a), \mathit{eff}(a))$, where $prec(a) \subseteq Lit(F)$ is the *precondition* of a and $\mathit{eff}(a)$ is a set of *conditional effects*, each of the form $C \to \ell$, where $C \subseteq Lit(F)$ is a *condition* and literal ℓ is the effect. Sometimes we write ℓ as a shorthand for the unconditional effect $\{\} \to \ell$.

We say that an action a is *applicable* in a planning state s iff $s \models prec(a)$. We denote by $\rho(s, a)$ the state that results from applying a in s. Formally,

$$\begin{aligned}\rho(s,a) =& (s \setminus \{p \mid C \to \neg p \in \mathit{eff}(a), s \models C\}) \cup \\ & \{p \mid C \to p \in \mathit{eff}(a), s \models C\}\end{aligned}$$

if $s \in F$ and a is applicable in s; otherwise, $\delta(a, s)$ is undefined. If α is a sequence of actions and a is an action, we define $\rho(s, \alpha a)$ as $\rho(\delta(s, \alpha), a)$ if $\rho(s, \alpha)$ is defined. Furthermore, if α is the empty sequence, then $\rho(s, \alpha) = s$.

An action sequence α is *applicable* in a state s iff $\rho(s, \alpha)$ is defined. If an action sequence $\alpha = a_1 a_2 \ldots a_n$ is applicable in s, it induces an execution trace $\sigma = s_1 \ldots s_{n+1}$ in s, where $s_i = \rho(I, a_1 \ldots a_{i-1})$, for every $i \in \{1, \ldots, n+1\}$.

An action sequence is a *plan* for problem $\langle F, O, I, G\rangle$ if α is applicable in I and $\rho(I, \alpha) \models G$.

2.3 Alternating Automata

Alternating automata (AA) are a natural generalization of non-deterministic finite-state automata (NFA). At a definitional level, the difference between an NFA and an AA is the transition function. For example, if A is an NFA with transition function δ, and we have that $\delta(q, a) = \{p, r\}$, then this intuitively means that A may end up in state p or in state r as a result of reading symbol a when A was previously in state q. With an AA, transitions are defined as formulas. For example, if δ' is the transition function for an AA A', then $\delta'(q, a) = p \vee r$ means, as before, that A' ends up in p or r after reading an a in state q. Nevertheless, formulas provide more expressive power. For example $\delta'(q, b) = (s \wedge t) \vee r$ can be intuitively understood as A' will end up in both s and t or (only) in r after reading a b in state q. In this model, only *positive Boolean formulas* are allowed for defining δ.

Definition 1 (Positive Boolean Formula) *The set of positive formulas over a set of propositions $\mathcal{P}$—denoted by $\mathcal{B}^+(\mathcal{P})$—is the set of all Boolean formulas over $\mathcal{P}$ and constants $\bot$ and $\top$ that do not use the connective "$\neg$".*

The formal definition for AA that we use henceforth follows.

Definition 2 (Alternating Automata) *An alternating automata (AA) over words is a tuple $A = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$, where Q is a finite set of states, Σ, the alphabet, is a finite set of symbols, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is the transition function, $\mathcal{I} \subseteq Q$ are the initial states, and $\mathcal{F} \subseteq Q$ is a set of final states.*

As suggested above, any NFA is also an AA. Indeed, given an NFA with transition function δ, we can generate an equivalent AA with transition function δ' by simply defining $\delta'(q, a) = \bigvee_{p \in P} p$, when $\delta(q, a) = P$. We observe that this means $\delta'(q, a) = \bot$ when P is empty.

As with NFAs, an AA accepts a word w whenever there exists a *run* of the AA over w that satisfies a certain property. Here is the most important (computational) difference between AAs and NFAs: a run of an AA is a sequence of *sets* of states rather than a sequence of states. Before defining runs formally, for notational convenience, we extend δ for any subset T of Q as $\delta(T, a) = \bigwedge_{q \in T} \delta(q, a)$ if $T \neq \emptyset$ and $\delta(T, a) = \top$ if $T = \emptyset$.

Definition 3 (Run of an AA over a Finite String) *A run of an AA $A = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ over word $x_1 x_2 \ldots x_n$ is a sequence $Q_0 Q_1 \ldots Q_n$ of subsets of Q, where $Q_0 = \mathcal{I}$, and $Q_i \models \delta(Q_{i-1}, x_i)$, for every $i \in \{1, \ldots, n\}$.*

Definition 4 *A word w is accepted by an AA A iff there is a run $Q_0 \ldots Q_n$ of A over w such that $Q_n \subseteq F$.*

For example, if the definition of an AA A is such that $\delta'(q, b) = (s \wedge t) \vee r$, and $\mathcal{I} = \{q\}$, then both $\{q\}\{s, t\}$ and $\{q\}\{r\}$ are runs of A over word b.

2.4 Finite LTL

The focus of this paper is planning with LTL interpreted over *finite* state sequences [Baier and McIlraith, 2006; De Giacomo and Vardi, 2013]. At the syntax level, the finite LTL we use in this paper is almost identical to regular LTL, except for the addition of a "weak next" modality ($\bullet$). The definition follows.

Definition 5 (Finite LTL formulas) *The set of finite LTL formulas over a set of propositions $\mathcal{P}$, fLTL($\mathcal{P}$), is inductively defined as follows:*

- *p is in fLTL($\mathcal{P}$), for every $p \in \mathcal{P}$.*

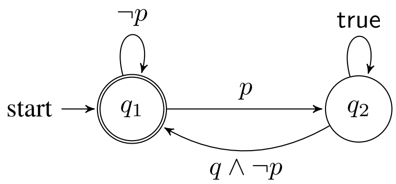

Figure 1: An NFA for formula $\Box(p \rightarrow \bigcirc\Diamond q)$ that expresses the fact that every time p becomes true in a state, then q has to be true in the state after or in the future. The input to the automaton is a (finite) sequence $s_0 \dots s_n$ of planning states.

- *If φ and ψ are in fLTL($\mathcal{P}$) then so are $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\bigcirc\varphi$, $\bullet\varphi$, $(\varphi \mathsf{U} \psi)$, and $(\varphi \mathsf{R} \psi)$.*

The truth value of a finite LTL formula is evaluated over a finite sequence of states. Below we assume that those states are actually planning states.

Definition 6 *Given a sequence of states $\sigma = s_0 \dots s_n$ and a formula $\varphi \in$ fLTL($\mathcal{P}$), we say that σ satisfies φ, denoted as $\sigma \models \varphi$, iff it holds that $\sigma, 0 \models \varphi$, where, for every $i \in \{0, \dots, n\}$:*

1. *$\sigma, i \models p$ iff $s_i \models p$, when $p \in \mathcal{P}$.*
2. *$\sigma, i \models \neg\varphi$ iff $\sigma, i \not\models \varphi$*
3. *$\sigma, i \models \psi \wedge \chi$ iff $\sigma, i \models \psi$ and $\sigma, i \models \chi$*
4. *$\sigma, i \models \psi \vee \chi$ iff $\sigma, i \models \psi$ or $\sigma, i \models \chi$*
5. *$\sigma, i \models \bigcirc\psi$ iff $i < n$ and $\sigma, (i+1) \models \psi$*
6. *$\sigma, i \models \bullet\psi$ iff $i = n$ or $\sigma, (i+1) \models \psi$*
7. *$\sigma, i \models \psi \mathsf{U} \chi$ iff there exists $k \in \{i, ..., n\}$ such that $\sigma, k \models \chi$ and for each $j \in \{i, ..., k-1\}$, it holds that $\sigma, j \models \psi$*
8. *$\sigma, i \models \psi \mathsf{R} \chi$ iff for each $k \in \{i, ..., n\}$ it holds that $\sigma, k \models \chi$ or there exists a $j \in \{i, ..., k-1\}$ such that $\sigma, j \models \psi$*

We sometimes use the macros $\mathsf{true} \stackrel{\text{def}}{=} p \vee \neg p$, $\mathsf{false} \stackrel{\text{def}}{=} \neg\mathsf{true}$, and $\varphi \rightarrow \psi$ as $\neg\varphi \vee \psi$. Additionally, $\Diamond\varphi$, pronounced as "eventually φ" is defined as $\mathsf{true} \,\mathsf{U}\, \varphi$, and $\Box\varphi$, pronounced as "always φ" is defined as $\neg\Diamond\neg\varphi$.

2.5 Deterministic Planning with LTL goals

A planning problem with a finite LTL goal is a tuple $P = \langle F, O, I, G \rangle$, where F, O, and I are defined as in classical planning problems, but where G is a formula in *fLTL*(F). An action sequence α is a plan for P if α is applicable in I, and the execution trace σ induced by the execution of α in I is such that $\sigma \models G$.

There are two approaches to compiling away LTL via non-deterministic finite-state automata [Edelkamp, Jabbar, and Naizih, 2006; Baier and McIlraith, 2006]. B&M's approach compiles away LTL formulas exploiting the fact that for every finite LTL formula φ it is possible to build an NFA that accepts the finite models of φ. To illustrate this, Figure 1 shows an NFA for $\Box(p \rightarrow \bigcirc\Diamond q)$. B&M represent the NFA within the planning domain using one fluent per automaton state. In the example of Figure 1, this means that the new planning problem contains fluents E_{q_1} and E_{q_2}. The translation is such that if α is a sequence of actions that induces the execution trace $\sigma = s_1 \dots s_n$, then E_q is true in s_n iff there is some run of the automaton over σ that ends in state q. B&M's translation has the following property.

Theorem 1 (Follows from [Baier, 2010]) *Let P be a classical planning problem, φ be a finite LTL formula, and P' be the instance that results from applying the B&M translation to P. Moreover, let α be a sequence of actions applicable in the initial state of P, and let σ be the sequence of (planning) states induced by the execution of α in P'. Finally, let A_φ be the NFA for φ. Then the following are equivalent statements.*

1. *There exists a run ρ of A_φ ending in q.*
2. *E_q is true in the last state of σ.*

As a corollary of the previous theorem, one obtains that satisfaction of finite LTL formulas can be determined by checking whether or not the disjunction $\bigvee_{f \in \mathcal{F}} E_f$ holds, where $\mathcal{F}$ denotes the set of final states of A_φ.

Unfortunately, B&M's translation is worst-case exponential [Baier, 2010]; for example, an NFA for $\wedge_{i=1}^{n} \Diamond p_i$ has 2^n states. Baier [2010] proposes a formula-partitioning technique that allows the method to generate more compact translations for certain formulas. The method, however, is not applicable to any formula.

Edelkamp's approach is similar to B&M's: it builds a Büchi automaton (BA), whose states are represented via fluents, compactly representing all runs of the automaton in a single planning state. The main difference is that the state of the automaton is updated via specific actions—a process that they call *synchronized update*. We modify this idea in the compilation we give below; however, our compilation is significantly different since it does not represent all runs of the automaton in the same planning state. It is important to remark that the use of BA interpreted as NFA does not yield a correct translation for general LTL goals, although it is correct for the PDDL3 subset of LTL [De Giacomo, Masellis, and Montali, 2014].

3 Alternating Automata and Finite LTL

A central part of our approach is the generation of an AA from an LTL formula. To do this we modify Muller, Saoudi, and Schupp's AA [1988] for *infinite* LTL formulas. Our AA is equivalent to a recent proposal by De Giacomo, Masellis, and Montali [2014]. The main difference between our construction and De Giacomo, Masellis, and Montali's is that we do not assume a distinguished proposition becomes true only in the final state. On the other hand, we require a special state (q_F) that indicates the sequence should finish. The use of such a state is the main difference between our AA for finite LTL and Muller, Saoudi, and Schupp's AA for infinite LTL.

We require the LTL input formula to be written in negation normal form (NNF); i.e., a form in which negations can be applied only to atomic formula. This transformation can be done in linear time [Gerth et al., 1995].

Let φ be in *fLTL*(S) and $sub(\varphi)$ be the set of the subformulas of φ, including φ. We define $A_\varphi = (Q, 2^S, \delta, q_\varphi, \{q_F\})$, where $Q = \{q_\alpha \mid \alpha \in sub(\varphi)\} \cup \{q_F\}$

and:

$$
\begin{aligned}
\delta(q_\ell, s) &= \begin{cases} \top, & \text{if } \ell \in Lit(F) \text{ and } s \models \ell \\ \bot, & \text{if } \ell \in Lit(F) \text{ and } s \not\models \ell \end{cases} \\
\delta(q_F, s) &= \bot \\
\delta(q_{\alpha\vee\beta}, s) &= \delta(q_\alpha, s) \vee \delta(q_\beta, s) \\
\delta(q_{\alpha\wedge\beta}, s) &= \delta(q_\alpha, s) \wedge \delta(q_\beta, s) \\
\delta(q_{\bigcirc\alpha}, s) &= q_\alpha \\
\delta(q_{\bullet\alpha}, s) &= q_F \vee q_\alpha \\
\delta(q_{\alpha \mathsf{U} \beta}, s) &= \delta(q_\beta, s) \vee (\delta(q_\alpha, s) \wedge q_{\alpha \mathsf{U} \beta}) \\
\delta(q_{\alpha \mathsf{R} \beta}, s) &= \delta(q_\beta, s) \wedge (q_F \vee \delta(q_\alpha, s) \vee q_{\alpha \mathsf{R} \beta})
\end{aligned}
$$

Theorem 2 *Given an LTL formula φ and a finite sequence of states σ, A_φ accepts σ iff $\sigma \models \varphi$.*

Proof sketch: Suppose that $\sigma = x_1 x_2 \dots x_n \in \Sigma^*$, where $\Sigma = 2^S$. The proof of the theorem is straightforward from the following lemma: φ: $\sigma, i \models \varphi$ if and only if there exists a sequence $r = Q_{i-1} Q_i \dots Q_n$, such that: (1) $Q_{i-1} = \{q_\varphi\}$, (2) $Q_n \subseteq \{q_F\}$, (3) For each subset Q_j in the sequence r it holds that $Q_j \subseteq sub(\varphi) \cup \{q_F\}$ and (4) For each $j \in \{i, i+1, \dots, n\}$ it holds that $Q_j \models \delta(Q_{j-1}, x_j)$. The proof for the lemma follows. It is inductive on the construction of φ.
$\Rightarrow$) Suppose that $\sigma, i \models \varphi$. To prove this direction, it suffices to provide a sequence $r = Q_{i-1} Q_i \dots Q_n$ satisfying the aforementioned properties. Below we show each sequence. We do not show that they satisfy the four properties; we leave this as an exercise to the reader.

- $\varphi = \ell$, for any literal ℓ. Then $r = (\{q_\ell\}, \emptyset, \dots, \emptyset)$.

Suppose that the lemma holds for any φ with less than m operators and that for any α and β with less than m operators, their respective sequences are $r' = Q'_{i-1} Q'_i Q''_{i+1} \dots Q'_n$ and $r'' = Q''_{i-1} Q''_i Q''_{i+1} \dots Q''_n$. Now, let φ be a formula with m operators:

- $\varphi = \alpha \vee \beta$. Then $\sigma, i \models \alpha$ or $\sigma, i \models \beta$. Without loss of generality, suppose that $\sigma, i \models \alpha$. Then $r = (\{q_\varphi\}, Q'_i, Q'_{i+1}, \dots, Q'_n)$.
- $\varphi = \alpha \wedge \beta$. Then $r = (\{q_\varphi\}, (Q'_i \cup Q''_i), (Q'_{i+1} \cup Q''_{i+1}), \dots, (Q'_n \cup Q''_n)$.
- $\varphi = \bigcirc\alpha$. Then $\sigma, (i+1) \models \alpha$. In this case, the sequence for α is $r' = Q'_i Q''_{i+1} \dots Q'_n$. With this, the sequence r for $\bigcirc\alpha$ is $r = (\{q_\varphi\}, \{q_\alpha\}, Q''_{i+1}, \dots, Q'_n)$.
- $\varphi = \bullet\alpha$. Then $i = n$ or $\sigma, (i+1) \models \alpha$. If $i = n$, the sequence $r = (Q_{n-1}, Q_n) = (\{q_\varphi\}, \{q_F\})$. If $i < n$, consider the same sequence r for the case $\bigcirc\alpha$.
- $\varphi = \alpha \mathsf{U} \beta$. Then, there exists $k \geq i$ such that $\sigma, k \models \beta$ and for every $j \in \{i, \dots, k-1\}$ it holds that $\sigma, j \models \alpha$. For β, assume its sequence is $r_k = (Q^k_{k-1}, Q^k_k, \dots, Q^k_n)$ and for each α that is satisfied by σ, j, assume its sequence is $r_j = (Q^j_{j-1}, Q^j_j, \dots, Q^j_n)$. The sequence $r = Q_{i-1} Q_i \dots Q_n$ is given by:

$$
Q_j = \begin{cases} \{q_{\alpha \mathsf{U} \beta}\}, \text{ if } j = i-1 \\ \{q_{\alpha \mathsf{U} \beta}\} \cup \bigcup_{x=i}^{j} Q^x_j, \text{ if } i-1 < j < k \\ \bigcup_{x=i}^{k} Q^x_j, \text{ if } j \geq k \end{cases}
$$

- $\varphi = \alpha \mathsf{R} \beta$. Then, for each $k \in \{i, \dots, n\}$ it holds that $\sigma, k \models \beta$ or there exists a $j \in \{i, \dots, k-1\}$ such that $\sigma, j \models \alpha$. If there is no such j, then $\sigma, k \models \beta$ for every $k \in \{i, \dots, n\}$ and for each one of them, assume their sequence will correspond to $r_k = (Q^k_{k-1}, Q^k_k, \dots, Q^k_n)$. The sequence $r = Q_{i-1} Q_i \dots Q_n$ is given by:

$$
Q_k = \begin{cases} \{q_{\alpha \mathsf{R} \beta}\}, \text{ if } k = i-1 \\ \{q_{\alpha \mathsf{R} \beta}\} \cup \bigcup_{x=i}^{k} Q^x_k, \text{ if } i-1 < k < n \\ \{q_F\}, \text{ if } k = n \end{cases}
$$

  If there is a $j \in \{i, \dots, k-1\}$ such that $\sigma, j \models \alpha$, consider the minimum such j and assume its sequence is $r' = (A_{j-1}, A_j, \dots, A_n)$. For $k \in \{i, \dots, j\}$, the sequences for β will be $r_k = (B^k_{k-1}, B^k_k, \dots, B^k_n)$. The sequence $r = Q_{i-1} Q_i \dots Q_n$ is given by:

$$
Q_k = \begin{cases} \{q_{\alpha \mathsf{R} \beta}\}, \text{ if } k = i-1 \\ \{q_{\alpha \mathsf{R} \beta}\} \cup \bigcup_{x=i}^{k} B^x_k, \text{ if } i-1 < k < j \\ A_k \cup \bigcup_{x=i}^{j} B^x_k, \text{ if } k \geq j \end{cases}
$$

$\Leftarrow$) Suppose that there exists a sequence $r = Q_{i-1} Q_i \dots Q_n$ for φ that satisfies the four properties. To prove that $\sigma, i \models \varphi$, it should be straightforward for $\varphi = \ell$. For the inductive steps, where α is a direct subformula of φ, the sequence r must be used to create a new sequence r' for α (ensuring that r' satisfies the four properties) and use the implication of $\sigma, i \models \alpha$. This finishes the proof for the lemma. ■

4 Compiling Away finite LTL Properties

Now we propose an approach to compiling away finite LTL properties using the AA construction described above.

First, we argue that the idea underlying both Edelkamp's and B&M's translations would *not* yield an efficient translation if applied to AA. Recall in both approaches if $E_{q_1}, \dots, E_{q_n}$ are true in a planning state s, then there are n runs of the automaton, each of which ends in $q_1, \dots, q_n$ (Theorem 1). In other words, the planning state keeps track of *all* of the runs of the automaton. To apply the same principle to AA, we would need to introduce one fluent for each *subset* of states of the AA, therefore generating a number of fluents exponential on the size of the original formula. This is because runs of AA are sequences of *sets* of states, so we would require states of the form E_R, where R is a set of states.

To produce an efficient translation, we renounce the idea of representing all runs of the automaton in a single planning state. Our translation will then only keep track of a *single* run.

4.1 Translating LTL via LTL Synchronization

Our compilation approach takes as input an LTL planning problem P and produces a new planning problem P', which is is like P but contains additional fluents and actions. Like previous compilations, A_G is represented in P' with additional fluents, one for each state of the automaton for G. Like in Edelkamp's compilation P' contains specific actions—below referred to as *synchronization actions*—whose only purpose is to update the truth values of those additional fluents. A plan for P' alternates one action from the original problem P with a number of synchronization actions. Unlike any other previous compilation, P' does not represent all possible runs of the automaton in a single planning state.

Synchronization actions update the state of the automaton following the definition of the δ function. The most notable characteristic that distinguishes synchronization from the Edelkamp' s translation is that non-determinism inherent to the AA is modeled using alternative actions, each of which represents the different non-deterministic options of the AA. As such if there are n possible non-deterministic choices, via the applications of synchronization actions there will be n reachable planning states, each representing a single run.

Given a planning problem $P = \langle F, O, I, G \rangle$, our translation generates a problem P' in which there is one (new) fluent q for each state q of the AA A_G. The compilation is such that the following property holds: if $\alpha = a_1 a_2 \ldots a_n$ is applicable in the initial state of P, then there exists a set $\mathcal{A}_\alpha$ of action sequences of the form $\alpha_0 a_1 \alpha_1 a_2 \alpha_2 \ldots a_n \alpha_n$, where each α_i is a sequence of synchronization actions whose sole objective is to update the fluents representing A_G's state.

Our theoretical result below says that our compilation can represent all runs, but only one run at a time. Specifically, each of the sequences of $\mathcal{A}_\alpha$ corresponds to some run of A_G over the state sequence induced by α over P. Moreover, if $\alpha' \in \mathcal{A}_\alpha$, E_q is true in the state resulting from performing sequence α' in P' iff q is *contained* in the last element of a run that corresponds to α'.

We we are ready to define P'. Assume the AA for G has the form $A_G = (Q, \Sigma, \delta, q_0, \{q_f\})$.

Fluents P' has the same fluents as P plus fluents for the representation of the states of the automaton (Q), flags for controlling the different modes ($\mathbf{copy}$, $\mathbf{sync}$, $\mathbf{world}$), and a special fluent $\mathbf{ok}$, which becomes false if the goal has been falsified. Finally, it includes the set $Q^S = \{q^S \mid q \in Q\}$ which are "copies" of the automata fluents, which we describe in detail below. Formally, $F' = F \cup Q \cup Q^S \cup \{\mathbf{copy}, \mathbf{sync}, \mathbf{world}, \mathbf{ok}\}$.

The set of operators O' is the union of the sets O_w and O_s.

World Mode Set O_w contains the same actions in O, but preconditions are modified to allow execution only in "world mode". Effects, on the other hand are modified to allow the execution of the *copy* action, which initiates the synchronization phase, and which is described below. Formally, $O_w = \{a' \mid a \in O\}$, and for all a' in O_w:

$$prec(a') = prec(a) \cup \{\mathbf{ok}, \mathbf{world}\},$$
$$\mathit{eff}(a') = \mathit{eff}(a) \cup \{\mathbf{copy}, \neg\mathbf{world}\}.$$

Synchronization Mode The synchronization mode can be divided in three consecutive phases. In the first phase, we execute the *copy* action which in the successor states adds a copy q^S for each fluent q that is currently true, deleting q. Intuitively, during synchronization, each q^S defines the state of the automaton prior to synchronization. The precondition of *copy* is simply $\{\mathbf{copy}, \mathbf{ok}\}$, while its effect is defined by:

$$\mathit{eff}(copy) = \{q \to q^S, q \to \neg q \mid q \in Q\} \cup \{\mathbf{sync}, \neg\mathbf{copy}\}$$

As soon as the $\mathbf{sync}$ fluent becomes true, the second phase of synchronization begins. Here the only executable actions are those that update the state of the automaton, which are defined in Table 1. Note that one of the actions deletes the $\mathbf{ok}$ fluent. This can happen, for example while synchronizing a formula that actually expresses the fact that the action sequence has to conclude now.

Sync Action	Precondition	Effect
$trans(q^S_\ell)$	$\{\mathbf{sync}, \mathbf{ok}, q^S_\ell, \ell\}$	$\{\neg q^S_\ell\}$
$trans(q^S_F)$	$\{\mathbf{sync}, \mathbf{ok}, q^S_F\}$	$\{\neg q^S_F, \neg\mathbf{ok}\}$
$trans(q^S_{\alpha\wedge\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\wedge\beta}\}$	$\{q^S_\alpha, q^S_\beta, \neg q^S_{\alpha\wedge\beta}\}$
$trans_1(q^S_{\alpha\vee\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\vee\beta}\}$	$\{q^S_\alpha, \neg q^S_{\alpha\vee\beta}\}$
$trans_2(q^S_{\alpha\vee\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\vee\beta}\}$	$\{q^S_\beta, \neg q^S_{\alpha\vee\beta}\}$
$trans(q^S_{\bigcirc\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\bigcirc\alpha}\}$	$\{q_\alpha, \neg q^S_{\bigcirc\alpha}\}$
$trans_1(q^S_{\bullet\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\bullet\alpha}\}$	$\{q_F, \neg q^S_{\bullet\alpha}\}$
$trans_2(q^S_{\bullet\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\bullet\alpha}\}$	$\{q_\alpha, \neg q^S_{\bullet\alpha}\}$
$trans_1(q^S_{\alpha \mathsf{U} \beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha \mathsf{U} \beta}\}$	$\{q^S_\beta, \neg q^S_{\alpha \mathsf{U} \beta}\}$
$trans_2(q^S_{\alpha \mathsf{U} \beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha \mathsf{U} \beta}\}$	$\{q^S_\alpha, q_{\alpha \mathsf{U} \beta}, \neg q^S_{\alpha \mathsf{U} \beta}\}$
$trans_1(q^S_{\alpha \mathsf{R} \beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha \mathsf{R} \beta}\}$	$\{q^S_\beta, q_F, \neg q^S_{\alpha \mathsf{R} \beta}\}$
$trans_2(q^S_{\alpha \mathsf{R} \beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha \mathsf{R} \beta}\}$	$\{q^S_\beta, q^S_\alpha, \neg q^S_{\alpha \mathsf{R} \beta}\}$
$trans_3(q^S_{\alpha \mathsf{R} \beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha \mathsf{R} \beta}\}$	$\{q^S_\beta, q_{\alpha \mathsf{R} \beta}, \neg q^S_{\alpha \mathsf{R} \beta}\}$

Table 1: The synchronization actions for LTL goal G in NNF. Above ℓ, $\alpha \mathsf{R} \beta$, $\alpha \mathsf{U} \beta$, and $\bigcirc\alpha$ are assumed to be in the set of subformulas of G. In addition, ℓ is assumed to be a literal.

When no more synchronization actions are possible, we enter the third phase of synchronization. Here only action $world$ is executable; its only objective is to reestablish world mode. The precondition of $world$ is $\{\mathbf{sync}, \mathbf{ok}\} \cup \overline{Q^S}$, and its effect is $\{\mathbf{world}, \neg\mathbf{sync}\}$.

The set O_s is defined as the one containing actions *copy*, $world$, and all actions defined in Table 1.

New Initial State The initial state of the original problem P intuitively needs to be "processed" by $\mathcal{A}_G$ before starting to plan. Therefore, we define I' as $I \cup \{q_G, \mathbf{copy}, \mathbf{ok}\}$.

New Goal Finally, the goal of the problem is to reach a state in which no state fluent in Q is true, except for q_f, which may be true. Therefore, we define $G' = \{\mathbf{world}, \mathbf{ok}\} \cup \overline{(Q \setminus \{q_F\})}$.

4.2 Properties

There are two important properties that can be proven about our translation. First, our translation is correct.

Theorem 3 (Correctness) *Let $P = \langle F, O, I, G \rangle$ be a planning problem with an LTL goal and $P' = \langle F', O', I', G' \rangle$ be the translated instance. Then P has a plan $a_1 a_2 \ldots a_n$ iff P' has a plan $\alpha_0 a_1 \alpha_1 a_2 \alpha_2 \ldots a_n \alpha_n$, in which for each $i \in \{0, \ldots, n\}$, α_i is a sequence of actions in O_s.*

Proof sketch: We show each sequence of actions α_i simulates the behavior of the automata, i.e., whenever t is a planning state whose next action must be *copy* and $q_\beta \in t$, then $\rho(t, \alpha_i)$ satisfies $\delta(q_\beta, t)$.

For this, let's define t^S as the subset of all the automata fluents Q^S that are added during the execution of the sequence of actions α_i. We will prove the following lemma by induction on the construction of φ: If $q^S_\varphi \in t^S$, then $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$:

Observe that if $q^S_\varphi \in t^S$, then there must be an action *trans*(q^S_φ) that was executed in α_i. This is because $\rho(t, \alpha_i) \cap Q^S = \emptyset$ and only *trans*(q^S_φ) can delete q^S_φ from the current state. The second observation is: If some action *trans* adds q^S_α, then $q^S_\alpha \in t^S$. This is by definition of t^S. If the action adds q_ψ, then $q_\psi \in \rho(t, \alpha_i)$, because the only action that deletes fluents in Q is *copy*.

- $\varphi = \ell$. Assume ℓ is positive literal. Then there is a planning state s in which $trans(q^S_\ell)$ was executed. Since the precondition requires $\ell \in s$ and ℓ can only be added by an action from O_w, then $\ell \in t$. By definition, $\delta(q_\ell, t) = \top$, and it is clear that $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$. The argument is analogous for a negative literal ℓ.

We will not consider the case for q_F. It is never desirable to synchronize that state, because the special fluent **ok** is removed, leading to a dead end. Now, assume that $q^S_\varphi \in t^S$ implies $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$ for every φ with less than m operators. The proof sketch for each case can be verified by the reader as follows:

- For each φ, it is clear that a version of $trans(q^S_\varphi)$ is executed due to the first observation.
- If q_ψ is added by *trans*, then $q_\psi \in \rho(t, \alpha_i)$ due to the second observation. This implies that $\rho(t, \alpha_i) \models q_\psi$.
- If q^S_α is added by *trans*, then $q^S_\alpha \in t^S$. By induction hypothesis, $\rho(t, \alpha_i) \models \delta(q_\alpha, t)$, because α is a strict subformula of φ and has less than m operators.
- Finally, using entailment (for positive boolean formulae) and the definition of the transitions for the alternating automata A_φ, it can be verified that $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$.
- The argument is similar for the other versions of *trans*.

To conclude our theorem, note that if t is a planning state, $q_\beta \in t$ and the next action to execute is *copy*, then $q^S_\beta \in t^S$. Using the lemma, this implies $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$. ■

Second, the size of the plan for P' is linear on the size of the plan for P.

Theorem 4 (Bounded synchronizations) *If T is a reachable planning state from I' and $T \cap Q^S \neq \emptyset$, then there is a sequence of trans actions σ such that $\delta(T, copy \cdot \sigma) \cap Q^S = \emptyset$ and $|\sigma| \in \mathcal{O}(|G|)$.*

Proof: Note that T is a state in *world* mode getting ready to go into *synchronization* mode after the *copy* action has been executed. The main idea of the proof is to choose the order of the subformulae to be synchronized, where the first one corresponds to the largest subformula of the current state, the second one corresponds to the second largest subformula and so on. Note that when an action $trans(q^S_\alpha)$ is executed, it always happens that at most two fluents q^S_β and q^S_γ are added, and the formulae β and γ are strict subformulae of α. This means that a subformula will never get synchronized twice in a single synchronization phase σ. Since the number of subformulae is linear on $|G|$, this means that the length of σ must be $O(|G|)$. ■

4.3 Towards More Efficient Translations

The translation we have presented above can be modified slightly for obtaining improved performance. The following are modifications that we have considered.

An Order for Synchronization Actions Consider the goal formula is $\alpha \wedge \beta$ and that currently both q_α and q_β are true. The planner has two equivalent ways of completing the synchronization: by executing first $trans(q_\alpha)$ and then $trans(q_\beta)$, or by inverting this sequence. By enforcing an order between these synchronizations, we can reduce the branching factor at synchronization phase. Such an order is simple to enforce by modifying preconditions and effects of synchronization actions so that states are synchronized following a topological order of the parse tree of G.

Positive Goals The goal condition of the translated instance requires being in and every $q \in Q$ to be false. On the other hand, action *copy*, which has to be performed after each world action, has precisely the effect of making every $q \in Q$ false. This may significantly hurt performance if search relies on heuristics that relax negative effects of actions, like the FF heuristic [Hoffmann and Nebel, 2001], which is key to the performance of state-of-the-art planning systems [Richter and Helmert, 2009]. To improve heuristic guidance, we define a new fluent q^D, for each $q \in Q$, with the intuitive meaning that q^D becomes true when $trans(q)$ cannot be executed in the future. For every action $trans(q^S_\alpha)$ that does not add q_α, we include the conditional effect $\{q_\beta \mid \beta \in super(\alpha)\} \rightarrow q^D_\alpha$, where $super(\alpha)$ is the set of subformulas of G that are proper superformulas of α. Using a function f that takes a $fLTL(F)$ formula and generates a propositional formula, the new goal $f(G)$ can be recursively written as follows:

- If $\varphi = p$ and $p \in Lit(F)$, then $f(p) = q^D_p$.
- If $\varphi = \alpha \wedge \beta$, then $f(\varphi) = q^D_\varphi \wedge f(\alpha) \wedge f(\beta)$
- If $\varphi = \alpha \vee \beta$, then $f(\varphi) = q^D_\varphi \wedge (f(\alpha) \vee f(\beta))$
- If $\varphi = \bigcirc\beta$ or $\varphi = \bullet\beta$, then $f(\varphi) = q^D_\varphi \wedge f(\beta)$
- If $\varphi = \alpha \star \beta$, where $\star \in \{\mathsf{U}, \mathsf{R}\}$, then $f(\varphi) = q^D_\varphi \wedge f(\beta)$

5 Empirical Evaluation

The objective of our evaluation was to compare our approach with existing translation approaches, over a range of general LTL goals, to understand when it is convenient to use one or other approach. We chose to compare to B&M's rather than Edelkamp's because the former seems to yield better performance [Baier, Bacchus, and McIlraith, 2009]. We do not compare against other existing systems that handle PDDL3 natively, such as LPRPG-P [Coles and Coles, 2011], because efficient translations for the (restricted) subset of LTL of PDDL3 into NFA are known [Gerevini et al., 2009].

We considered both LAMA [Richter, Helmert, and Westphal, 2008] and FF$_\mathcal{X}$ [Thiébaux, Hoffmann, and Nebel, 2005], because both are modern planners supporting derived predicates (required by B&M). We observed that LAMA's preprocessing times where high, sometimes exceeding planning time by 1 to 2 orders of magnitude, and thus decided to report results we obtained with FF$_\mathcal{X}$. We used an 800MHz-CPU machine running Linux. Processes were limited to 1 GB of RAM and 15 min. runtime.

There are no planning benchmarks with general LTL goals, so we chose two of the domains (rovers and openstacks) of the 2006 International Planning Competition, which included

	B&M's translator				Non-PG + Non-OSA					Non-PG + OSA					PG + Non-OSA					PG + OSA				
	TT	PL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS
Openstacks Domain																								
a03	0.373	23	0.10	34	0.463	136	21	6.44	222245	0.486	309	21	10.86	203461	0.475	167	23	0.05	1413	0.501	319	22	0.25	3421
a04	1.594	0	NR	NR	0.470	156	22	22.49	592081	0.504	392	22	24.04	417585	0.496	192	24	0.10	2763	0.527	405	23	0.52	6573
a05	21.852	0	NR	NR	0.482	179	23	103.30	1573433	0.523	481	23	54.07	872446	0.525	213	24	0.23	5906	0.564	497	24	1.17	13074
e03	0.377	23	0.10	34	0.459	117	21	9.08	294097	0.481	287	21	10.57	202873	0.469	167	23	0.05	1413	0.491	297	22	0.23	3217
e04	1.599	0	NR	NR	0.472	125	22	31.93	755539	0.498	369	22	23.41	418240	0.489	192	24	0.10	2763	0.517	382	23	0.48	6176
e05	22.390	0	NR	NR	0.478	133	23	149.88	1958261	0.518	457	23	53.27	876816	0.513	213	24	0.22	5906	0.548	473	24	1.05	12292
f03	0.246	23	0.02	34	0.455	142	21	5.41	196938	0.474	265	21	8.78	179157	0.462	166	23	0.05	1412	0.483	274	22	0.21	3100
f05	0.268	25	0.02	36	0.477	199	23	78.71	1364638	0.503	433	23	48.62	834783	0.504	212	24	0.22	5905	0.536	448	24	1.01	12172
g02	0.256	24	0.10	214	0.466	166	21	20.15	533753	0.491	331	21	18.44	341998	0.484	197	22	0.41	10439	0.509	342	22	0.48	6498
g03	0.266	24	0.13	224	0.474	215	22	138.11	1892750	0.510	415	22	35.84	635715	0.506	231	22	1.53	35117	0.542	410	22	1.07	13364
h01	2.423	2	96.98	3	0.474	21	2	0.41	8907	0.504	49	2	0.95	14569	0.495	19	2	0.00	86	0.529	44	2	0.04	498
h02	4.567	2	0.00	3	0.477	20	2	2.58	51965	0.510	52	2	0.67	10967	0.505	32	2	0.07	1634	0.535	48	2	0.07	1002
Rovers Domain																								
e03	0.407	10	0.05	16	0.481	62	10	68.39	1173227	0.507	144	10	46.04	544034	0.498	67	10	0.02	453	0.517	140	10	0.02	448
e04	1.639	0	NR	NR	0.494	0	0	NR	NR	0.539	1	0	NR	NR	0.517	106	15	0.05	1324	0.545	252	15	0.07	1201
f01	0.242	4	0.00	5	0.460	25	4	0.03	1757	0.471	31	4	0.04	1353	0.462	22	4	0.00	64	0.469	28	4	0.00	62
f02	0.255	7	0.00	10	0.468	45	7	0.55	25490	0.487	73	7	1.84	44002	0.475	42	7	0.01	316	0.489	69	7	0.01	181
f03	0.270	10	0.01	15	0.481	68	10	7.67	264568	0.501	133	10	41.78	522432	0.490	66	10	0.02	452	0.505	129	10	0.03	425
i04	0.299	3	0.01	4	0.499	19	2	0.04	1522	0.528	49	2	0.04	847	0.525	17	3	0.00	43	0.548	47	3	0.02	301
j04	0.301	3	0.01	4	0.501	19	2	0.03	1290	0.537	49	2	0.05	962	0.522	26	5	0.01	148	0.554	78	5	0.20	3494
Blocksworld Domain																								
a03	1.627	2	0.02	3	0.448	22	2	0.12	4115	0.472	46	2	0.04	792	0.464	19	2	0.00	32	0.490	41	2	0.01	146
a04	22.220	0	NR	NR	0.458	25	2	1.94	45113	0.492	55	2	0.31	4692	0.486	22	2	0.00	50	0.523	50	2	0.01	228
a05	471.574	0	NR	NR	0.471	28	2	38.99	474906	0.514	64	2	2.52	24761	0.522	25	2	0.01	77	0.559	59	2	0.04	330
b03	9.801	1	0.00	2	0.446	11	1	0.00	88	0.473	31	1	0.01	122	0.464	8	1	0.00	12	0.494	25	1	0.00	85
b04	423.327	1	0.00	2	0.463	11	1	0.01	172	0.494	37	1	0.02	248	0.490	8	1	0.00	16	0.523	31	1	0.01	147
b05	NR	NR	NR	NR	0.473	11	1	0.02	285	0.519	43	1	0.05	492	0.527	8	1	0.00	22	0.566	37	1	0.02	269
c03	0.383	2	0.58	4	0.443	24	2	0.09	3607	0.470	46	2	0.01	357	0.465	22	2	0.00	101	0.489	41	2	0.01	97
c04	1.809	0	NR	NR	0.456	27	2	1.80	44666	0.490	55	2	0.06	1067	0.487	25	2	0.00	194	0.519	50	2	0.02	162
c05	25.655	0	NR	NR	0.470	30	2	48.30	610049	0.509	64	2	0.31	3325	0.520	28	2	0.04	514	0.558	59	2	0.04	287
d03	0.234	2	0.00	3	0.452	24	2	0.11	3451	0.481	49	2	0.07	1612	0.470	21	2	0.00	34	0.502	43	2	0.00	106
d04	0.243	2	0.01	3	0.467	28	2	2.66	48436	0.508	61	2	1.08	14118	0.513	25	2	0.00	53	0.550	55	2	0.02	172
d05	0.251	2	0.01	3	0.486	32	2	84.40	637178	0.535	73	2	17.99	104398	0.566	29	2	0.01	81	0.609	67	2	0.04	256
e03	3.813	0	NR	NR	0.457	25	2	0.16	5680	0.489	52	2	0.09	1700	0.483	22	2	0.00	35	0.517	46	2	0.00	150
e04	182.181	0	NR	NR	0.476	29	2	6.16	122233	0.517	64	2	1.15	14576	0.532	26	2	0.01	54	0.567	58	2	0.03	257
e05	NR	NR	NR	NR	0.495	0	0	NR	NR	0.547	76	2	19.13	106729	0.584	30	2	0.01	82	0.638	70	2	0.06	396

Table 2: Experimental results for a variety of LTL planning tasks.

LTL preferences (but not goals), and generated our own problems, with some of our goals inspired by the preferences. In addition, we considered the blocksworld domain.

Our translator was implemented in SWI-Prolog. It takes a domain and a problem in PDDL with an LTL goal as input and generates PDDL domain and problem files. It also receives an additional parameter specifying the translation mode which can be any of the following: *simple*, *OSA*, *PG*, and *OSA+PG*, where *simple* is the translation of Section 4, and *OSA*, *PG* are the optimizations described in Section 4.3. *OSA+PG* is the combination of *OSA* and *PG*.

Table 2 shows a representative selection of the results we obtained. It shows translation time (TT), plan length (PL), planning time (PT), the number of planning states that were evaluated before the goal was reached (PS). Times are displayed in seconds. For our translators we also include the length of the plan without synchronization actions (WPL). NR means the planner/translator did not return a plan. For each problem, a special name of the form $x0n$ was assigned, where x corresponds to a specific family of formula and n its parameter (i.e. For the problem $a04$, the goal formula $\alpha \cup \bigwedge_{i=1}^{n} \beta_i$ was used, with $n = 4$).

Each family of formulae corresponds to: a: $\alpha \cup \bigwedge_{i=1}^{n} \beta_i$, b: $\bigvee_{i=1}^{n} \Diamond p_i \,\mathsf{U}\, \Diamond q$, c: $\Diamond(\alpha \wedge \bigwedge_{i=1}^{n} \Diamond \beta_i)$, d: $\bigwedge_{i=1}^{n}(\alpha \,\mathsf{U}\, \beta_i)$, e: $\Diamond(\bigwedge_{i=1}^{n} \Diamond \beta_i)$, f: $\bigwedge_{i=1}^{n} \Diamond p_i$, g: $\bigwedge_{i=1}^{3} \Diamond p_i \wedge \bigwedge_{i=1}^{n-3} q_i \,\mathsf{U}\, r_i$, h: $\alpha \,\mathsf{U}\, \beta$, where α or β has n nested operators U or R, i: $\Diamond(\bigvee_{i=1}^{n} \Diamond \beta_i)$ and j: $\Diamond(\bigvee_{i=1}^{n} \Box \beta_i)$.

We observe mixed results. B&M yields superior results on some problems; e.g., $f03$ and $f05$ of *openstacks* (of the form $\bigwedge_{i=1}^{n} \Diamond p_i$). The performance gap is probably due to the fact that (1) the B&M problem requires fewer actions in the plan and (2) B&M's output for these goals is quite compact on the size of the formula. On the other hand, there are other goal formulas in which our approach outperforms B&M. For example, problems of the form $a0n$ in openstacks and blocksworld, and of the form $b0n$ in blocksworld. In those cases, the B&M translator is forced to generate the whole automaton, because it has to deal with nested subformulae in which the distributive property does not hold. As a consequence, B&M generates an output exponential in n, which results in higher translation time and eventually in the the planner running out of memory.

By observing the rest of the data, we conclude that B&M returns an output that is significantly larger than our approaches for the following classes of formulas: $\alpha \,\mathsf{U}(\bigwedge_{i=1}^{n} \Diamond \beta_i)$, $\alpha \,\mathsf{U}(\bigwedge_{i=1}^{n} \beta_i \,\mathsf{U}\, \gamma_i)$, $(\bigvee_{i=1}^{n} \Box \alpha_i) \,\mathsf{U}\, \beta$, and $(\bigvee_{i=1}^{n} \alpha_i \,\mathsf{R}\, \beta_i) \,\mathsf{U}\, \beta$, with $n \geq 4$, yielding finally an "NR". Being polynomial, our translation handles these formulas reasonably well: low translation times, and a compact output. In many cases, this allows the planner to return a solution.

The use of positive goals has an important influence in performance possibly because the heuristic is more accurate, leading to fewer expansions. OSA, on the other hand, seems to negatively affect planning performance in $\mathrm{FF}_{\mathcal{X}}$. The reason is the following: $\mathrm{FF}_{\mathcal{X}}$ will frequently choose the wrong synchronization action and therefore its enforced hill climbing algorithm will often fail. This behavior may not be observed in planners that use complete search algorithms.

6 Conclusions

We proposed polynomial-time translations of LTL into final-state goals, which, unlike existing translations are optimal with respect to computational complexity. The main difference between our approach and state-of-the-art NFA-based translations is that we use AA, and represent a single run of the AA in the planning state. We conclude from our experimental data that it seems more convenient to use an our AA translation precisely when the output generated by the NFA-based translation is exponentially large in the size of the formula. Otherwise, it seems that NFA-based translations are more efficient because they do not require synchronization actions, which require longer plans, and possibly higher planning times. Obviously, a combination of both translation approaches into one single translator should be possible. Investigating such a combination is left for future work.

Acknowledgements

We thank the anonymous reviewers and acknowledge funding from the *Millennium Nucleus Center for Semantic Web Research* under Grant NC120004, and from Fondecyt Grant 1150328, both from the Republic of Chile.

References

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

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

[Baier and McIlraith, 2006] Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 788–795.

[Baier, Bacchus, and McIlraith, 2009] Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 173(5-6):593–618.

[Baier, 2010] Baier, J. A. 2010. *Effective Search Techniques for Non-Classical Planning via Reformulation*. Ph.D. in Computer Science, University of Toronto.

[Bylander, 1994] Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1-2):165–204.

[Coles and Coles, 2011] Coles, A. J., and Coles, A. 2011. LPRPG-P: relaxed plan heuristics for planning with preferences. In *Proceedings of the 21th International Conference on Automated Planning and Scheduling*.

[Cresswell and Coddington, 2004] Cresswell, S., and Coddington, A. M. 2004. Compilation of LTL goal formulas into PDDL. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th European Conference on Artificial Intelligence*, 985–986. Valencia, Spain: IOS Press.

[De Giacomo and Vardi, 1999] De Giacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In Biundo, S., and Fox, M., eds., *ECP*, volume 1809 of *LNCS*, 226–238. Durham, UK: Springer.

[De Giacomo and Vardi, 2013] De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces.

[De Giacomo, Masellis, and Montali, 2014] De Giacomo, G.; Masellis, R. D.; and Montali, M. 2014. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 1027–1033.

[Edelkamp, Jabbar, and Naizih, 2006] Edelkamp, S.; Jabbar, S.; and Naizih, M. 2006. Large-scale optimal PDDL3 planning with MIPS-XXL. In *5th International Planning Competition Booklet*, 28–30.

[Edelkamp, 2006] Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling*.

[Gerevini et al., 2009] Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

[Gerth et al., 1995] Gerth, R.; Peled, D.; Vardi, M. Y.; and Wolper, P. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification*, 3–18.

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

[Muller, Saoudi, and Schupp, 1988] Muller, D. E.; Saoudi, A.; and Schupp, P. E. 1988. Weak Alternating Automata Give a Simple Explanation of Why Most Temporal and Dynamic Logics are Decidable in Exponential Time. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science*, 422–427.

[Pnueli, 1977] Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, 46–57.

[Richter and Helmert, 2009] Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*.

[Richter, Helmert, and Westphal, 2008] Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 975–982.

[Rintanen, 2000] Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In Horn, W., ed., *Proceedings of the 14th European Conference on Artificial Intelligence*, 526–530. Berlin, Germany: IOS Press.

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