Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces

Alberto Camacho,[†] Eleni Triantafillou,[†] Christian Muise,[⋆] Jorge A. Baier,[‡] Sheila A. McIlraith[†]
[†]Department of Computer Science, University of Toronto
[⋆]CSAIL, Massachusetts Institute of Technology
[‡]Center for Semantic Web Research, Pontificia Universidad Católica de Chile
[†]{acamacho,eleni,sheila}@cs.toronto.edu, [⋆]cjmuise@mit.edu, [‡]jabaier@ing.puc.cl

Abstract

Temporally extended goals are critical to the specification of a diversity of real-world planning problems. Here we examine the problem of non-deterministic planning with temporally extended goals specified in linear temporal logic (LTL), interpreted over *either* finite or infinite traces. Unlike existing LTL planners, we place no restrictions on our LTL formulae beyond those necessary to distinguish finite from infinite interpretations. We generate plans by compiling LTL temporally extended goals into problem instances described in the Planning Domain Definition Language that are solved by a state-of-the-art fully observable non-deterministic planner. We propose several different compilations based on translations of LTL to alternating or non-deterministic (Büchi) automata, and evaluate various properties of the competing approaches. We address a diverse spectrum of LTL planning problems that, to this point, had not been solvable using AI planning techniques, and do so in a manner that demonstrates highly competitive performance.

1 Introduction

Real-world planning problems involve complex goals that are temporally extended, require adherence to safety and liveness constraints, and often necessitate the optimization of preferences or other quality measures. Linear Temporal Logic (LTL) is a language that can be used to specify such constraints (Pnueli 1977).

Planning with *deterministic* actions and LTL goals has been well studied, commencing with the works of Bacchus and Kabanza (2000) and Doherty and Kvarnström (2001). Significant attention has been given to compilation-based approaches (e.g., (Rintanen 2000; Cresswell and Coddington 2004; Edelkamp 2006; Baier and McIlraith 2006; Patrizi et al. 2011)), which take a planning problem with an LTL goal and transform it into a classical planning problem for which state-of-the-art classical planning technology can often be leveraged. The more challenging problem of planning with *non-deterministic* actions and LTL goals has not been studied to the same extent; Kabanza, Barbeau, and St.-Denis (1997), and Pistore and Traverso (2001) have proposed their own LTL planners, while Patrizi, Lipovetzky, and Geffner (2013) have proposed the only compilation-based approach that existed prior to this work. Unfortunately, the latter approach is limited to the proper subset of LTL for which there exists deterministic Büchi automata. In addition, it is restricted to the interpretation of LTL over *infinite* traces and the compilation is worst-case exponential in the size of the goal formula. Finally, it is subject to a double-exponential blowup, since there exist LTL formulae of size n for which any recognizing *deterministic* Büchi automaton has 2^{2^n} states (Kupferman and Rosenberg 2010).

In this paper, we propose four compilation-based approaches to LTL planning with non-deterministic actions that exploit translations from LTL to alternating or non-deterministic (Büchi) automata. In each case, we compile LTL goals, together with the original planning domain, specified in the Planning Domain Definition Language (PDDL), into PDDL that is suitable for input to standard fully observable non-deterministic (FOND) planners. We compare compilations that exploit alternating automata to those that exploit non-deterministic automata, in the differing cases where LTL is interpreted over either finite or infinite traces. The approaches based on non-deterministic automata compilations capture the full expressivity of LTL for finite and infinite traces. They are sound, complete, and although the compilations are theoretically worst-case exponential in size with respect to the size of the original LTL, they do not manifest these exponential properties in practice. The other two approaches based on alternating automata have compilations that are linear in the size of the original LTL. The approaches are sound but incomplete for both the finite and infinite cases. To obtain completeness we would have lost the linear properties of the compilation. This restricted linear compilation produced inferior performance compare to our non-deterministic approach, so we did not include our complete alternating automata based approaches which lose the linear properties whose merits we wished to evaluate.

Our approaches build on methods for finite LTL planning with deterministic actions by Baier and McIlraith (2006) and Torres and Baier (2015), and for the infinite non-deterministic case, on the work of Patrizi, Lipovetzky, and Geffner (2013). While in the finite case the adaptation of these methods was fairly straightforward, the infinite case required non-trivial insights and modifications to Torres and Baier's and Baier and McIlraith's approaches. We evaluate

the relative performance of our compilation-based systems using a state-of-the-art FOND planner, and demonstrate that they are competitive with or superior to the state of the art.

Our work presents the first realization of a compilation-based approach to planning with *non-deterministic* actions where the LTL is interpreted over finite traces. Furthermore, unlike previous approaches to LTL planning, our compilations make it possible, for the first time, to solve the full spectrum of FOND planning with LTL goals interpreted over infinite traces. Table 1 summarizes existing compilation-based approaches and the contributions of this work. Importantly, our compilations can be seen as a practical step towards the efficient realization of a class of LTL synthesis tasks using planning technology (e.g., (Pnueli and Rosner 1989; De Giacomo and Vardi 2015)). We elaborate further with respect to related work in Section 5.

2 Preliminaries

2.1 FOND Planning

Following Ghallab, Nau, and Traverso (2004), a *Fully Observable Non-Deterministic* (FOND) planning problem is a tuple $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$, where $\mathcal{F}$ is a set of *fluents*; $\mathcal{I} \subseteq \mathcal{F}$ characterizes what holds initially; $\mathcal{G} \subseteq \mathcal{F}$ characterizes the goal; and $\mathcal{A}$ is the set of actions. The set of literals of $\mathcal{F}$ is $Lits(\mathcal{F}) = \mathcal{F} \cup \{\neg f \mid f \in \mathcal{F}\}$. Each action $a \in \mathcal{A}$ is associated with $\langle Pre_a, \mathit{Eff}_a \rangle$, where $Pre_a \subseteq Lits(\mathcal{F})$ is the precondition and Eff_a is a set of outcomes of a. Each outcome $e \in \mathit{Eff}_a$ is a set of conditional effects (with, possibly, an empty condition), each of the form $(C \rightarrow \ell)$, where $C \subseteq Lits(\mathcal{F})$ and $\ell \in Lits(\mathcal{F})$. Given a planning state $s \subseteq \mathcal{F}$ and a fluent $f \in \mathcal{F}$, we say that s satisfies f, denoted $s \models f$ iff $f \in s$. In addition $s \models \neg f$ if $f \notin s$, and $s \models L$ for a set of literals L, if $s \models \ell$ for every $\ell \in L$. Action a is *applicable* in state s if $s \models Pre_a$. We say s' *is a result of applying* a *in* s iff, for one outcome e in Eff_a, s' is equal to $s \setminus \{p \mid (C \rightarrow \neg p) \in e, s \models C\} \cup \{p \mid (C \rightarrow p) \in e, s \models C\}$. The *determinization* of a FOND problem $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ is problem $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A}' \rangle$, where each non-deterministic action $a \in \mathcal{A}$ is replaced by a set of deterministic actions, a_i, one action corresponding to each of the distinct non-deterministic effects of a.

Solutions to a FOND planning problem $\mathcal{P}$ are *policies*. A policy p is a partial function from states to actions such that if $p(s) = a$, then a is applicable in s. An *execution* of a policy p in state s is an infinite sequence $(s_0, a_0), (s_1, a_1), \ldots$ or a finite sequence $(s_0, a_0), \ldots, (s_{n-1}, a_{n-1}), s_n$, where $s_0 = s$, and all of its state-action-state substrings s, a, s' are such that $p(s) = a$ and s' is a result of applying a in s. Finite executions ending in a state s are such that $p(s)$ is undefined. An execution σ *yields* the state trace π that results from removing all the action symbols from σ. Alternatively, solutions to $\mathcal{P}$ can be represented as *finite-state controllers* (FSCs). We refer the reader to (Geffner and Bonet 2013; Patrizi, Lipovetzky, and Geffner 2013) for details.

Following Geffner and Bonet (2013), an infinite execution σ is *fair* iff whenever s, a occurs infinitely often within σ, then so does s, a, s', for every s' that is a result of applying a in s. A solution to $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ is *strong cyclic* iff each of its executions in $\mathcal{I}$ is either finite and ends in a state that satisfies $\mathcal{G}$ or is (infinite and) unfair. Intuitively, such fairness over executions of a strong-cyclic solution guarantees that from every reachable state, a goal state can eventually be reached assuming that no effect is always avoided.

2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) was first proposed for verification (Pnueli 1977). An LTL formula is interpreted over an infinite sequence, or *trace*, of states. LTL interpreted over *finite* traces has received attention from the planning community (Gerevini and Long 2005; Baier and McIlraith 2006). De Giacomo and Vardi (2013) provided a formal description and named it $\mathsf{LTL_f}$. LTL and $\mathsf{LTL_f}$ use modal operators *next* ($\bigcirc$), and *until* (U), from which it is possible to define the well-known operators *always* ($\Box$) *eventually* ($\Diamond$), and *release* (R). $\mathsf{LTL_f}$, in addition, allows a *weak next* ($\bullet$) operator. An LTL/ $\mathsf{LTL_f}$ formula over a set of propositions $\mathcal{P}$ is defined inductively: a proposition in $\mathcal{P}$ is a formula, and if ψ and χ are formulae, then so are $\neg\psi$, $(\psi \wedge \chi)$, $(\psi \,\mathsf{U}\, \chi)$, $\bigcirc\psi$, and $\bullet\psi$.

The semantics of LTL and $\mathsf{LTL_f}$ is defined as follows. Formally, a state trace π is a sequence of states, where each state is an element in $2^{\mathcal{P}}$. We assume that the first state in π is s_1, that the i-th state of π is s_i and that $|\pi|$ is the length of π (which is ∞ if π is infinite). We say that π satisfies φ ($\pi \models \varphi$, for short) iff $\pi, 1 \models \varphi$, where for every $i \geq 1$:

- $\pi, i \models p$, for a propositional variable $p \in \mathcal{P}$, iff $p \in s_i$,
- $\pi, i \models \neg\psi$ iff it is not the case that $\pi, i \models \psi$,
- $\pi, i \models (\psi \wedge \chi)$ iff $\pi, i \models \psi$ and $\pi, i \models \chi$,
- $\pi, i \models \bigcirc\varphi$ iff $i < |\pi|$ and $\pi, i+1 \models \varphi$,
- $\pi, i \models (\varphi_1 \,\mathsf{U}\, \varphi_2)$ iff for some j in $\{i, \ldots, |\pi|\}$, it holds that $\pi, j \models \varphi_2$ and for all $k \in \{i, \ldots, j-1\}$, $\pi, k \models \varphi_1$,
- $\pi, i \models \bullet\varphi$ iff $i = |\pi|$ or $\pi, i+1 \models \varphi$.

Observe operator $\bullet$ is equivalent to $\bigcirc$ iff π is infinite. As such we allow $\bullet$ in LTL formulae, we do not use the acronym $\mathsf{LTL_f}$, but we are explicit regarding which interpretation we use (either finite or infinite) when not obvious from the context. As usual, $\Diamond\varphi$ is defined as $(\mathsf{true}\,\mathsf{U}\,\varphi)$, $\Box\varphi$ as $\neg\Diamond\neg\varphi$, and $(\psi \,\mathsf{R}\, \chi)$ as $\neg(\neg\psi \,\mathsf{U}\, \neg\chi)$.

Given an LTL/$\mathsf{LTL_f}$ formula φ there exists an automaton $\mathcal{A}_\varphi$ that accepts a trace π iff $\pi \models \varphi$. Depending on whether π is finite or infinite, different types of automata are needed.

For the finite case, such automata may be either non-deterministic or alternating. A finite trace is accepting when a run of the automaton finishes in a so-called *accepting* automaton state. For the infinite case, such automata may be either Büchi non-deterministic or Büchi alternating (Vardi and Wolper 1994). Büchi automata accept an infinite trace when a run of the automaton visits accepting automaton states infinitely often. Alternation generates compact automata, the number of states in $\mathcal{A}_\varphi$ is linear in the size of φ (in both the infinite and finite cases), while non-deterministic (Büchi) automata are worst-case exponential.

These automata constructions can be exploited to compile LTL goals into planning problems that use existing planning technology for non-temporal goals. A challenge to

Infinite LTL		Finite LTL	
Deterministic Actions	Non-Deterministic Actions	Deterministic Actions	Non-Deterministic Actions
[Albarghouthi et al., 2009] (EXP) [Patrizi et al., 2011] (EXP)	[Patrizi et al., 2013] (limited LTL) (2EXP) **[this paper (ABA)]** (limited LTL) (LIN) **[this paper (NBA)]** (EXP)	[Edelkamp, 2006] (EXP) [Cresswell & Coddington, 2006] (EXP) [Baier & McIlraith, 2006] (EXP) [Torres & Baier, 2015] (LIN)	**[this paper (AA)]** (limited LTL) (LIN) **[this paper (NFA)]** (EXP)

Table 1: Automata-based compilations for LTL planning. In brackets, size of resulting automata relative to the original LTL formula: (double-)exponential, (2)EXP; linear, LIN. Our approaches are based on non-deterministic finite state automata, NFA; non-deterministic Büchi automata , NBA; alternating automata, AA; and alternating Büchi automata, ABA.

planning with LTL is that LTL imposes a path constraint on the definition of a valid plan. Highly optimized techniques for heuristic search planning work well when the goal is a final-state goal. As such the idea behind a compilation approach to planning with LTL is to transform LTL temporally extended goal formulae into one or more automata. The original planning problem is then augmented to include the automata accepting conditions as final-state goals, and the initial state of the automata as part of the initial state of the planning problem. The transition system of the original problem is similarly augmented so that the automata states are updated as additional effects of the planning domain actions. Table 1 summarizes state-of-the-art automata-based approaches for deterministic and FOND LTL planning. As noted in the introduction, there has been very little work on non-deterministic planning with LTL.

3 FOND Planning with LTL Goals

An LTL-FOND problem is a tuple $\langle \mathcal{F}, \mathcal{I}, \varphi, \mathcal{A} \rangle$, where $\mathcal{F}$, $\mathcal{I}$, and $\mathcal{A}$ are defined as in FOND problems, and φ is an LTL formula. Solutions to an LTL-FOND problems are FSCs.

Definition 1 (Finite LTL-FOND)**.** *An FSC* Π *is a solution for* $\langle \mathcal{F}, \mathcal{I}, \varphi, \mathcal{A} \rangle$ *under the finite semantics iff every execution of* Π *over* $\mathcal{I}$ *is such that either (1) it is finite and yields a state trace* π *such that* $\pi \models \varphi$ *or (2) it is (infinite and) unfair.*

Definition 2 (Infinite LTL-FOND)**.** *An FSC* Π *is a solution for* $\langle \mathcal{F}, \mathcal{I}, \varphi, \mathcal{A} \rangle$ *under the infinite semantics iff (1) every execution of* Π *over* $\mathcal{I}$ *is infinite and (2) every fair (infinite) execution yields a state trace* π *such that* $\pi \models \varphi$*.*

Our general approach to solve an LTL-FOND problem, $\mathcal{P}$, comprises three steps. In step one, $\mathcal{P}$ is compiled into a FOND problem $\mathcal{P}'$. In step two, a sound and complete FOND planner is used to produce a policy that solves the compiled problem. Finally, in step three, the resultant policy is transformed into an FSC that is a solution to $\mathcal{P}$. Below we present four compilations – two for Infinite LTL-FOND (Section 3.1), and two for Finite LTL-FOND (Section 3.2). Each exploits a correspondence between LTL and either alternating or non-deterministic automata.

We refer to step one of our approach as the *compilation* step and to the three steps collectively as the *AA-based approach* (respectively, *NFA-based*, *ABA-based*, *NBA-based*), depending on which automata are employed in the compilation step. The step three transformation of the policy into the FSC is straightforward, simply involving removal of all mention of the automata (extra bookkeeping fluents and actions we introduce to track automata state) from the step-two policy to produce a final FSC. The interested reader can find the details in (Camacho et al. 2016).

Our approaches are the first to address the full spectrum of FOND planning with LTL goals interpreted over finite and infinite traces. In particular our work is the first to solve *full* LTL-FOND with respect to infinite trace interpretations, and represents the first realization of a compilation approach for LTL-FOND with respect to finite trace interpretations.

3.1 From Infinite LTL-FOND to FOND

We present two different compilations for Infinite LTL-FOND. The first exploits alternating Büchi automata (ABA) and is linear in time and space with respect to the size of the LTL formula. The second exploits non-deterministic Büchi automata (NBA), and is worst-case exponential in time and space with respect to the size of the LTL formula. Interestingly, the NBA-based approach's performance is superior.

3.1.1 ABA-based Compilation Our ABA-based compilation builds on ideas by Torres and Baier (2015) for alternating automata (AA) based compilation of *finite* LTL planning with *deterministic* actions (henceforth, TB15), and from Patrizi, Lipovetzky, and Geffner's compilation (2013) (henceforth, PLG13) of LTL-FOND to FOND. Combining these two approaches is not straightforward. Among other reasons, TB15 does not yield a sound compilation for the infinite case, and thus we needed to modify it significantly. This is because the accepting condition for ABAs is more involved than that of regular AAs.

Given an LTL-FOND problem, the first step in the compilation is to build an ABA for our LTL goal formula φ over propositions $\mathcal{F}$, which we henceforth assume to be in negation normal form (NNF). Transforming an LTL formula φ to NNF can be done in linear time in the size of φ. The ABA we use below is an adaptation of the ABA by Vardi (1995). It is represented by a tuple $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_\varphi, Q_{Fin})$, where the set of states, $Q = \{q_\alpha \mid \alpha \in sub(\varphi)\}$, Σ contains all sets of propositions in $\mathcal{P}$, $Q_{Fin} = \{q_{\alpha \,\mathsf{R}\, \beta} \mid \alpha \,\mathsf{R}\, \beta \in sub(\varphi)\}$ and the transition function, δ is given by:

$$\delta(q_\ell, s) = \begin{cases} \top & \text{if } s \models \ell \text{ } (\ell \text{, literal}) \\ \bot & \text{otherwise} \end{cases}$$
$$\delta(q_{\alpha \wedge \beta}, s) = \delta(q_\alpha, s) \wedge \delta(q_\beta, s)$$
$$\delta(q_{\bigcirc \alpha}, s) = q_\alpha$$
$$\delta(q_{\alpha \vee \beta}, s) = \delta(q_\alpha, s) \vee \delta(q_\beta, s)$$
$$\delta(q_{\alpha \,\mathsf{U}\, \beta}, s) = \delta(q_\beta, s) \vee (\delta(q_\alpha, s) \wedge q_{\alpha \,\mathsf{U}\, \beta})$$
$$\delta(q_{\alpha \,\mathsf{R}\, \beta}, s) = \delta(q_\beta, s) \wedge (\delta(q_\alpha, s) \vee q_{\alpha \,\mathsf{R}\, \beta})$$

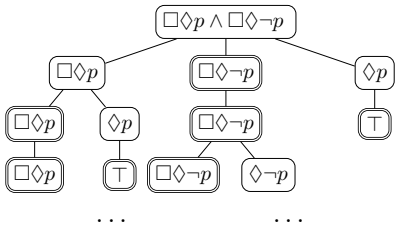


Figure 1: An accepting run of an ABA for $\Box\Diamond p \wedge \Box\Diamond\neg p$ over an infinite sequence of states in which the truth value of p alternates. Double-line ovals are accepting states/conditions.

For the reader unfamiliar with ABAs, the transition function for these automata takes a state and a symbol and returns a positive Boolean formula over the set of states Q[1]. Furthermore, a *run* of an ABA over an infinite string $\pi = s_1 s_2 \dots$ is characterized by a tree with labeled nodes, in which (informally): (1) the root node is labeled with the initial state, (2) level i corresponds to the processing of symbol s_i, and (3) the children of a node labeled by q at level i are the states appearing in a minimal model of $\delta(q, s_i)$. As such, multiple runs for a certain infinite string are produced when selecting different models of $\delta(q, s_i)$. A special case is when $\delta(q, s_i)$ reduces to $\top$ or $\bot$, where there is one child labeled by $\top$ or $\bot$, respectively. A run of an ABA is *accepting* iff all of its finite branches end on $\top$ and in each of its infinite branches there is an accepting state that repeats infinitely often. Figure 1 shows a run of the ABA for $\Box\Diamond p \wedge \Box\Diamond\neg p$—a formula whose semantics forces an infinite alternation, which is not necessarily immediate, between states that satisfy p and states that do not satisfy p.

Our ABA compilation for LTL-FOND follows a similar approach to that developed by TB15: given an input problem $\mathcal{P}$, we generate an equivalent problem $\mathcal{P}'$ in which we represent the configuration of the ABA with fluents (one fluent q per each state q of the ABA). $\mathcal{P}'$ contains the actions in $\mathcal{P}$ plus *synchronization actions* whose objective is to update the configuration of the ABA. In $\mathcal{P}'$, special fluents alternate between so-called *world mode*, in which only one action of $\mathcal{P}$ is allowed, and *synchronization mode*, in which the configuration of the ABA is updated.

Before providing details of the compilation we overview the main differences between our compilation and that of TB15. TB15 recognizes an accepting run (i.e., a satisfied goal) by observing that all automaton states at the last level of the (finite) run are accepting states. In the infinite case, such a check does not work. As can be seen in the example of Figure 1, there is no single level of the (infinite) run that contains only ABA accepting states.

Thus, when building a plan with our compilation, the planner "decides" at any moment that an accepting run can be found and then the objective is to "prove" this is the case by showing the existence of a loop or *lasso* in the plan in which any non-accepting state may turn into an accepting state. To keep track of those non-accepting states that we require to eventually "turn into" accepting states we use special fluents that we call *tokens*.

[1]To simplify the explanation we refer to formulae, φ, rather than q_φ as in our definition of δ.

Sync Action	Effect
$tr(q^S_\ell)$	$\{\neg q^S_\ell, q^T_\ell \rightarrow \neg q^T_\ell\}$
$tr(q^S_{\alpha\wedge\beta})$	$\{q^S_\alpha, q^S_\beta, \neg q^S_{\alpha\wedge\beta}, q^T_{\alpha\wedge\beta} \rightarrow \{q^T_\alpha, q^T_\beta, \neg q^T_{\alpha\wedge\beta}\}\}$
$tr_1(q^S_{\alpha\vee\beta})$	$\{q^S_\alpha, \neg q^S_{\alpha\vee\beta}, q^T_{\alpha\vee\beta} \rightarrow \{q^T_\alpha, \neg q^T_{\alpha\vee\beta}\}\}$
$tr_2(q^S_{\alpha\vee\beta})$	$\{q^S_\beta, \neg q^S_{\alpha\vee\beta}, q^T_{\alpha\vee\beta} \rightarrow \{q^T_\beta, \neg q^T_{\alpha\vee\beta}\}\}$
$tr(q^S_{\bigcirc\alpha})$	$\{q_\alpha, \neg q^S_{\bigcirc\alpha}, q^T_{\bigcirc\alpha} \rightarrow \{q^T_\alpha, \neg q^T_{\bigcirc\alpha}\}\}$
$tr_1(q^S_{\alpha\,\mathsf{U}\,\beta})$	$\{q^S_\beta, \neg q^S_{\alpha\,\mathsf{U}\,\beta}, q^T_{\alpha\,\mathsf{U}\,\beta} \rightarrow \{q^T_\beta, \neg q^T_{\alpha\,\mathsf{U}\,\beta}\}\}$
$tr_2(q^S_{\alpha\,\mathsf{U}\,\beta})$	$\{q^S_\alpha, q_{\alpha\,\mathsf{U}\,\beta}, \neg q^S_{\alpha\,\mathsf{U}\,\beta}, q^T_{\alpha\,\mathsf{U}\,\beta} \rightarrow q^T_\alpha\}$
$tr_1(q^S_{\alpha\,\mathsf{R}\,\beta})$	$\{q^S_\beta, q^S_\alpha, \neg q^S_{\alpha\,\mathsf{R}\,\beta}, q^T_{\alpha\,\mathsf{R}\,\beta} \rightarrow \neg q^T_{\alpha\,\mathsf{R}\,\beta}\}$
$tr_2(q^S_{\alpha\,\mathsf{R}\,\beta})$	$\{q^S_\beta, q_{\alpha\,\mathsf{R}\,\beta}, \neg q^S_{\alpha\,\mathsf{R}\,\beta}, q^T_{\alpha\,\mathsf{R}\,\beta} \rightarrow \neg q^T_{\alpha\,\mathsf{R}\,\beta}\}$
$tr_1(q^S_{\Diamond\alpha})$	$\{q^S_\alpha, \neg q^S_{\Diamond\alpha}, q^T_{\Diamond\alpha} \rightarrow \{q^T_\alpha, \neg q^T_{\Diamond\alpha}\}\}$
$tr_2(q^S_{\Diamond\alpha})$	$\{q_{\Diamond\alpha}, \neg q^S_{\Diamond\alpha}\}$
$tr(q^S_{\Box\alpha})$	$\{q^S_\alpha, q_{\Box\alpha}, \neg q^S_{\Box\alpha}, q^T_{\Box\alpha} \rightarrow \neg q^T_{\Box\alpha}\}$

Table 2: Synchronization actions. The precondition of $tr(q^S_\psi)$ is $\{\mathbf{sync}, q^S_\psi\}$, plus ℓ when $\psi = \ell$ is a literal.

For an LTL-FOND problem $\mathcal{P} = \langle \mathcal{F}, \mathcal{I}, \varphi, \mathcal{A} \rangle$, where φ is an NNF LTL formula with ABA $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_\varphi, Q_{Fin})$, the compiled FOND problem is $\mathcal{P}' = \langle \mathcal{F}', \mathcal{I}', \mathcal{G}', \mathcal{A}' \rangle$, where each component is described below.

Fluents and Actions $\mathcal{P}'$ has the same fluents as $\mathcal{P}$ plus fluents to represent the states of automata $F_Q = \{q_\psi \mid q_\psi \in Q\}$, and flags $\mathbf{copy}$, $\mathbf{sync}$, $\mathbf{world}$ for controlling the different modes. It also includes the set $F^S_Q = \{q^S_\psi \mid q_\psi \in Q\}$ which are *copies* of the automata fluents, and *tokens* F^T_Q=$\{q^T_\psi \mid q_\psi \in Q\}$. We describe both sets below. Formally, $F' = F \cup F_Q \cup F^S_Q \cup F^T_Q \cup \{\mathbf{copy}, \mathbf{sync}, \mathbf{world}, \mathbf{goal}\}$. The set of actions $\mathcal{A}'$ is the union of the sets $\mathcal{A}_w$ and $\mathcal{A}_s$ plus the *continue* action.

World Mode $\mathcal{A}_w$ contains the actions in $\mathcal{A}$ with preconditions modified to allow execution only in *world* mode. Effects are modified to allow the execution of the *copy* action, which initiates the synchronization phase, described below. Formally, $\mathcal{A}_w = \{a' \mid a \in \mathcal{A}\}$, and for all a' in $\mathcal{A}_w$ $Pre_{a'} = Pre_a \cup \{\mathbf{world}\}$, and $\mathit{Eff}_{a'} = \mathit{Eff}_a \cup \{\mathbf{copy}, \neg\mathbf{world}\}$.

Synchronization Mode This mode has three phases. In the first phase, the *copy* action is executed, adding a copy q^S for each fluent q that is currently true, deleting q. Intuitively, q^S defines the state of the automaton prior to synchronization. The precondition of *copy* is $\{\mathbf{copy}\}$, while its effect is: $\{q \rightarrow \{q^S, \neg q\} \mid q \in F_Q\} \cup \{\mathbf{sync}, \neg\mathbf{copy}\}$.

When 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, as defined in Table 2. These actions update the state of the automaton following the definition of the transition function, δ.

Additionally, each synchronization action for ψ with an associated token q^T_ψ, *propagates* such a token to its subformulae, unless ψ corresponds to either an accepting state (i.e., ψ is of the form $\alpha\,\mathsf{R}\,\beta$) or to a literal ℓ whose truth can be verified with respect to the current state via action $tr(q^S_\ell)$.

When no more synchronization actions are possible, we enter the third phase of synchronization. Here only two ac-

tions are executable: $world$ and $continue$. The objective of the $world$ action is to reestablish world mode. Its precondition is $\{\mathbf{sync}\} \cup \{\neg q^S \,|q^S \in F_Q^S\}$, and its effect is $\{\mathbf{world}, \neg\mathbf{sync}\}$.

The $continue$ action also reestablishes world mode, but in addition "decides" that an accepting ABA can be reached in the future. This is reflected by the non-deterministic effect that makes the fluent $\mathbf{goal}$ true. As such, it "tokenizes" all states that are not final states in F_Q, by adding q^T for each ABA state q that is non-final and currently true. Formally, the precondition of $continue$ is $\{\mathbf{sync}\} \cup \{\neg q_\varphi^T \mid \varphi \notin Q_{Fin}\}$, and its effect is $\{\{\mathbf{goal}\}, \{q_\varphi \rightarrow q_\varphi^T \mid \varphi \notin Q_{Fin}\} \cup \{\mathbf{world}, \neg\mathbf{sync}\}\}$. The set $\mathcal{A}_s$ contains the actions $copy$, $world$, and all actions defined in Table 2.

Initial and Goal States The resulting problem $\mathcal{P}'$ has initial state $I' = I \cup \{q_\varphi, \mathbf{copy}\}$, and goal $\mathcal{G}' = \{\mathbf{goal}\}$.

Our ABA-based compilation builds on TB15 while integrating ideas from PLG13. Like PLG13 our compilation uses a $continue$ action to find plans with lassos, but unlike PLG13, our compilation does not directly use the accepting configuration of the automaton. Rather, the planner "guesses" that such a configuration can be reached. The token fluents F_Q^T, which did not exist in TB15, are created for each non-accepting state and can only be eliminated when a non-accepting ABA state becomes accepting.

The ABA-based compilation produces a FOND problem, $\mathcal{P}'$. It is straightforward to generate an FSC for $\mathcal{P}$ from a strong cyclic policy for $\mathcal{P}'$ via simulation, removing explicit mention of automata fluents and synchronization actions. following our description in (Camacho et al. 2016).

Throughout the paper, soundness guarantees that FSCs obtained from solutions to the compiled problem $\mathcal{P}'$ are solutions to the LTL-FOND problem $\mathcal{P}$. Similarly, completeness guarantees that if a solution exists for $\mathcal{P}$, then one exists for $\mathcal{P}'$. Following (Torres and Baier 2015), Theorem 1 establishes that the size of the ABA-based compilation is linear in the size of the LTL formula. Linear size is preserved at the expense of losing completeness of the ABA-based approach.

Theorem 1. *The ABA-based compilation for Infinite* LTL*-FOND is linear in the size of the goal formula.*

Theorem 2. *The ABA-based approach to Infinite* LTL*-FOND is sound.*

Proof sketch. A policy p' for $\mathcal{P}'$ yields three types of executions: (1) finite executions that end in a state where $\mathbf{goal}$ is true, (2) infinite executions in which the $continue$ action is executed infinitely often and (3) infinite, unfair executions. We do not need to consider (3) because of Definition 2. Because the precondition of $continue$ does not admit token fluents, if $continue$ executes infinitely often we can guarantee that any state that was not an ABA accepting state turns into an accepting state. This in turn means that every branch of the run contains an infinite repetition of final states. Soundness of our approach reuses most of the argument that TB15 uses to show their compilation is sound, and follows from the argument above, the soundness of the FOND planner, and the straightforward construction of the FSC. □

As we can see in the definition of δ, the progression of subformuae ψ of the form $\alpha \vee \beta$, $\alpha \,\mathsf{U}\, \beta$, and $\alpha \,\mathsf{R}\, \beta$ involves a disjunction of subformulae. In our compilation, the synchronization actions make a deterministic choice, and *commit* to satisfy one of the disjuncts in the progression of the automata fluent q_ψ, while forgetting about the other disjunct. The advantage of making these deterministic choices is that the size of the compilation becomes worst-case linear in the size of the formula. The downside, however, is that the compilation is not always complete – although soundness is maintained – because the progressions of the subformulae may not capture all accepting runs of the automaton.

By way of illustration, consider a problem with unique action, a, with no preconditions, and non-deterministic effects that either make p true or false. Consider the subformula $\psi = \bigcirc p \vee \bigcirc\neg p$. The progression of ψ, following Table 2, either commits to make p true in the next time step, or to make $\neg p$ true in the next time step. Despite the fact that ψ is a tautology and there exists a trivial strategy that satisfies ψ, the compiled problem has no solution. Intuitively, the completeness guarantees of the compilation is lost when the part that is forgotten talks about future time steps. However, our compilations are complete when either $\psi = \alpha \vee \beta$ and one of α or β is a literal, or $\psi = \alpha \,\mathsf{U}\, \beta$ and both α and β are literals (similarly with a release operator).

Theorem 3. *The ABA-based approach to Infinite* LTL*-FOND is complete when, for all subformulae ψ in $sub(\varphi)$:*

- *if $\psi = \alpha \vee \beta$, then either α or β are formulae that do not contain temporal operators*
- *if $\psi = \alpha \,\mathsf{U}\, \beta$, or $\psi = \alpha \,\mathsf{R}\, \beta$ then α and β are formulae that do not contain temporal operators*

Proof sketch. When the disjuncts of the progression of ψ do not contain temporal operators, the truth of the deterministic choice made by synchronization actions is evaluated in the current state and does not depend on any effect of future actions. This makes it possible to consume subformulae of the type $\psi = \alpha \,\mathsf{U}\, \beta$ and $\psi = \alpha \,\mathsf{R}\, \beta$. If $\psi = \alpha \vee \beta$, it suffices that one of α, or β does not contain temporal operators. The completeness of our approach reuses most of the argument that TB15 uses to show their compilation is complete, and follows from the discussion above, the completeness of the FOND planner, and the construction of the FSC. □

3.1.2 NBA-based Compilation Our NBA-based compilation relies on the construction of a non-deterministic Büchi automaton (NBA) for the goal formula, and builds on compilation techniques for *finite* LTL planning with *deterministic* actions developed by Baier and McIlraith (2006) (henceforth, BM06). Given a deterministic planning problem $\mathcal{P}$ with LTL goal φ, the BM06 compilation runs in two phases: first, φ is transformed into a non-deterministic finite-state automata (NFA), $\mathcal{A}_\varphi$, such that it accepts a finite sequence of states σ if and only if $\sigma \models \varphi$. In the second phase, it builds an output problem $\mathcal{P}'$ that contains the same fluents as in $\mathcal{P}$ plus additional fluents of the form F_q, for each state q of $\mathcal{A}_\varphi$. Problem $\mathcal{P}'$ contains the same actions as in $\mathcal{P}$ but each action may contain additional effects which model the dynamics of the F_q fluents. The goal of $\mathcal{P}'$ is defined as the disjunction of all fluents of the form F_f, where f is an accepting state of $\mathcal{A}_\varphi$. The initial state of $\mathcal{P}'$ contains F_q iff

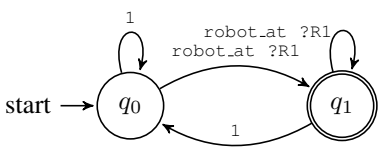


Figure 2: NBA of the LTL formula $\Box\Diamond$`robot_at ?R1`.

q is a state that $\mathcal{A}_\varphi$ would reach after processing the initial state of $\mathcal{P}$. The most important property of BM06 is the following: let $\sigma = s_0 s_1 \dots s_{n+1}$ be a state trace induced by some sequence of actions $a_0 a_1 \dots a_n$ in $\mathcal{P}'$, then F_q is satisfied by s_{n+1} iff there exists a run of $\mathcal{A}_\varphi$ over σ that ends in q. This means that a single sequence of planning states encodes *all* runs of the NFA $\mathcal{A}_\varphi$. The important consequence of this property is that the angelic semantics of $\mathcal{A}_\varphi$ is immediately reflected in the planning states and does not need to be handled by the planner (unlike TB15).

For an LTL-FOND problem $\mathcal{P} = \langle \mathcal{F}, \mathcal{I}, \varphi, \mathcal{A} \rangle$, our NBA-based compilation constructs a FOND problem $\mathcal{P}' = \langle \mathcal{F}', \mathcal{I}', \mathcal{G}', \mathcal{A}' \rangle$ via the following three phases: (i) construct an NBA, $\mathcal{A}_\varphi$ for the NNF LTL goal formula φ, (ii) apply the *modified* BM06 compilation to the determinization of $\mathcal{P}$ (see Section 2.1), and (iii) construct the final FOND problem $\mathcal{P}'$ by undoing the determinization, i.e., reconstruct the non-deterministic actions from their determinized counterparts.

As an illustrative example, consider the problem where a robot that navigates between rooms has to patrol room `R1` infinitely often. The associated LTL goal formula is $\Box\Diamond$`robot_at ?R1`, which compiles into the NBA in Figure 2. The robot is initially in room `R0`. It can perform the action *move* to attempt to move to a desired room. In case of failure, the robot stays where it is. Figure 3 shows the two deterministic actions, *move_eff1* and *move_eff2*, that result from the BM06 compilation of the determinization of the action *move*. Both (parametrized) actions have the same preconditions. The angelic non-determinism of the automaton is encoded within the dynamics of the actions, so that transitions capture all runs of the automaton. The non-determinism of the environment can be reconstructed by joining the two deterministic actions into a single, non-deterministic action with the same precondition and two non-deterministic effects: the effects of actions *move_eff1* and *move_eff2*.

The modification of the BM06 compilation used in the second phase leverages ideas present in PLG13 and our ABA-based compilations to capture infinite runs via induced non-determinism. The NFA-based compilation uses $continue\_q$ actions, one for each accepting automata fluent F_q, whose precondition is $\{F_q\}$. As with the ABA-based compilations, the $continue\_q$ actions have one non-deterministic effect that achieves $\mathbf{goal}$, while the other forces the search for infinite plans by requiring the planner to perform at least one *world* action. This is ensured by adding an extra precondition to $continue\_q$, $\mathbf{can\_continue}$, which is true in the initial state, is made true by every action but $continue\_q$, and is deleted by $continue\_q$. The other effect deletes all the automata fluents except q. I.e., $\mathit{Eff}_{continue\_q} = \{\{\mathbf{goal}\}, \{F_{q'} \rightarrow \neg F_{q'} \mid q' \neq q\} \cup \{\mathbf{can\_continue}\}\}$.

```
(:action move_eff1
  :parameters (?from ?to)
  :precondition (robot_at ?from)
  :effect (and (robot_at ?to) (not (robot_at ?from))
    (when (and (autstate_q0) (or (= ?to R1)
      (and (robot_at R1) (not (= ?from R1)))))
      (autstate_q1))
    (when (or (not (autstate_q0)) (and (not (= ?to R1))
      (or (not (robot_at R1)) (= ?from R1))))
      (not (autstate_q1)))))

(:action move_eff2
  :parameters (?from ?to)
  :precondition (robot_at ?from)
  :effect (and
    (when (and (autstate_q0) (robot_at R1))
      (autstate_q1))
    (when (or (not (autstate_q0)) (not (robot_at R1))
      (not (autstate_q1))))))
```

Figure 3: BM06 compilation of the determinization of the action *move* of the Robot example.

Unlike the ABA-based compilation, the tokenization is not required because accepting runs are those that achieve accepting states infinitely often, no matter which ones. However, the NFA-based compilations require a set of $continue\_q$ actions. One might be tempted to define a unique $continue$ action, instead, whose precondition is the accepting configuration of the NBA (a disjunction of the fluents representing accepting states), and whose effect is $\mathit{Eff}_{continue} = \{\{\mathbf{goal}\}, \{\mathbf{can\_continue}\}\}$. This will not produce sound solutions. In this case, it is true that for each accepting state reached during execution of a strong cyclic policy, there exists a run of the automata that reaches it. However, there is no guarantee of existence of *one* run of the automata that reaches infinite accepting states.

Theorem 4. *The NBA-based compilation for* Infinite LTL-*FOND planning is worst-case exponential in the size of the goal formula.*

Theorem 5. *The NBA-based approach to* Infinite LTL-*FOND planning is sound and complete.*

The transformation of LTL into NBA is worst-case exponential. Theorems 4 and 5 follow from soundness and completeness of the BM06 compilation, this time using an NBA automaton, and an argument similar to that of Theorem 2. This time, if $continue\_q$ actions execute infinitely often we can guarantee accepting NBA states are reached infinitely often. Similar to the AA-based approach, an FSC can be easily constructed from a strong-cyclic solution to $\mathcal{P}$.

3.2 From Finite LTL-FOND to FOND

Our compilations for Finite LTL-FOND extend the BM06 and TB15 compilations, originally designed for *finite* LTL planning with *deterministic* actions, to the non-deterministic action setting. Both the original BM06 and TB15 compilations share two general steps. In step one, the LTL goal formula is transformed into an automaton/automata – in the case of BM06 an NFA, in the case of TB15, an AA. In step

two, a planning problem $\mathcal{P}'$ is constructed by augmenting $\mathcal{P}$ with additional fluents and action effects to account for the integration of automata. In the case of BM06, these capture the automata state and how domain actions update automata state. In the case of TB15, $\mathcal{P}$ must also be augmented with synchronization actions. In both compilations, problem goals are augmented with automata accepting states.

For LTL-FOND, BM06 and TB15 must be modified to account for non-deterministic action effects. We do so in a manner similar to the previous NBA- and ABA-based compilations. In particular, the LTL-FOND problem is determinized, the BM06 (resp. TB15) compilation is applied to the determinized problem, and then the non-deterministic actions are reconstructed from their determinized counterparts (as done in the NBA-based compilation) to produce the FOND problem, $\mathcal{P}'$. An FSC solution, Π, to the LTL-FOND problem $\mathcal{P}$, can be obtained from a solution to $\mathcal{P}'$.

The size of the NFA- and AA-based compilations follow, respectively, from the sizes of the BM06 and TB15 compilations. Likewise, soundness and completeness of the compilation-based approaches follow from the soundness and completeness of BM06 and TB15, and similar arguments applied in the proofs of Theorems 2, 3, and 5.

Theorem 6. *The NFA-based (resp. AA-based) compilation for Finite* LTL*-FOND is exponential (resp. linear) in the size of the goal formula.*

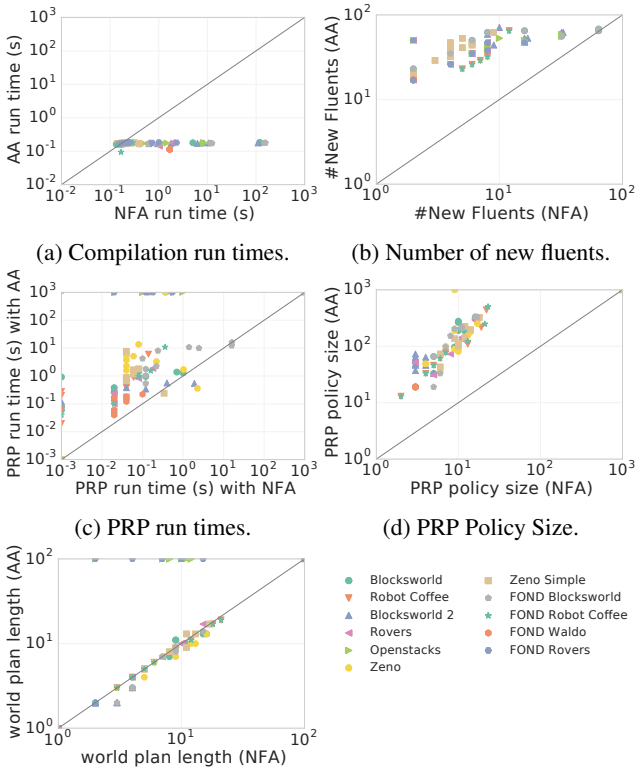
Theorem 7. *The NFA-based approach to Finite* LTL*-FOND is sound and complete.*

Theorem 8. *The AA-based approach to Finite* LTL*-FOND is sound, and is complete under the Theorem 3 conditions.*

4 Experiments

We evaluated our approaches to LTL-FOND planning in a selection of benchmarks with LTL goals from (Baier and McIlraith 2006; Patrizi, Lipovetzky, and Geffner 2013; Torres and Baier 2015), modified to include non-deterministic actions. We used PRP (Muise, McIlraith, and Beck 2012) as the FOND planner. Experiments were conducted on an Intel Xeon E5-2430 machine running at 2.2GHz, using a 4GB memory limit and a 30-min timeout.

LTL-FOND Planning over Finite Traces We evaluated the performance of our NFA- and AA-based planning systems, with respect to a collection of problems with deterministic and non-determinisitic actions and LTL goals, interpreted on finite traces. NFA-based compilation run times increased when the LTL formula had a large number of conjunctions and nested modal operators, whereas AA-based compilation times remain negligible. However, the compilations with AA included a number of new fluents that were, in some cases, up to one order of magnitude larger than with NFA (Figures 4a and 4b). Problems compiled with AA are, in general, more difficult to solve (Figure 4c). Because this compilation is incomplete, some problems are unsolvable. We report these cases as timeouts, even though PRP was able to detect unsolvability at pre-process time. AA-based compilations introduce a number of synchronization actions, resulting in greater policies than those obtained with



(a) Compilation run times. (b) Number of new fluents.

(c) PRP run times. (d) PRP Policy Size.

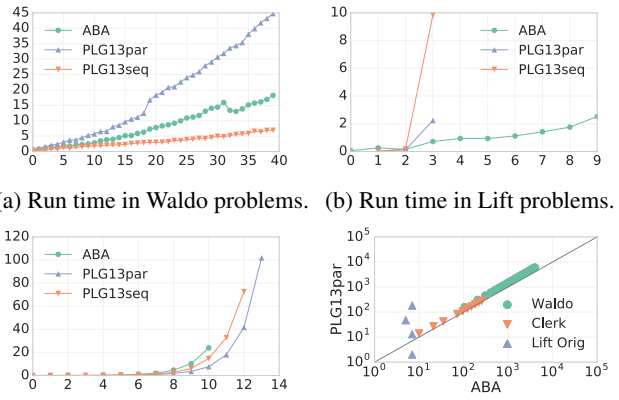
(e) Preferred *world* Plan Length.

Figure 4: Performance of our planning system using AA- and NFA-based compilations in problems with deterministic and non-deterministic actions and *finite* LTL goals. The NFA-based approach demonstrates superior performance.

NFA (Figure 5d). When these actions are excluded from the count, as per counting the size of the resulting FSCs, we get similar numbers. As a crude estimate of plan quality, we compared the number of *world* actions (i.e., excluding automaton-state *synchronization* actions) in the shortest plans of the policies (Figure 4e). The number of world actions with this metric was very similar in both compilations.

Interestingly, despite the AA-based compilations are not complete in theory, they were able to solve the majority of problems with a variety of LTL goal formulae. In addition, despite the size of the AA-based compilations is linear in the size of the original LTL formula and NFA compilations are worst-case exponential, in practice we observed the size of the NFA-based compiled problems is smaller. Further, PRP performs better when problems are compiled using NFAs, generating similar quality policies in lower search run times. We did not experience any decrease in performance in deterministic problems that were extended with non-deterministic actions, suggesting that AA- and NFA-based compilations remain competitive in LTL-FOND.

LTL-FOND Planning over Infinite Traces The relative performance observed between our NBA- and ABA-based approaches for LTL-FOND planning, interpreted over infinite traces, is reflective of the finite case. For reference,



(a) Run time in Waldo problems. (b) Run time in Lift problems.

(c) Run time in Clerk problems. (d) Policy Size (world actions).

Figure 5: Performance of our planning system using ABA-based compilations in problems with non-deterministic actions and *infinite* LTL goals. Our system demonstrates comparable performance with the PLG13 approach.

we compared the performance of FOND planning in our ABA-based compilations with the so-called *sequential* and *parallel* compilations developed by Patrizi, Lipovetzky, and Geffner (2013), subsequently referred to as PLG13seq and PLG13par, respectively. The former alternates between world and sync actions (that update the automaton state), whereas the latter parallelizes this process in a single action. The current implementation of PLG13 compilations forced us to limit the comparison to the three domains that appear in (Patrizi, Lipovetzky, and Geffner 2013). Namely, the *Waldo*, *Lift*, and *Clerk* domains. All problems have LTL goals that can be compiled into deterministic Büchi automata.

The results of experiments are summarized in Figure 5. We report PRP run times (sec) and policy sizes, excluding synchronization actions. In *Waldo* problems, the planner run times using ABA-based compilations are situated between the run times with PLG13seq and PLG13par. In *Lift* problems, the ABA compilations demonstrate significantly greater scalability. The *Lift* problems contain a (increasing) large number of conjunctive LTL goals. We conjecture that the poor scalability with PLG13seq (runs out of time) and PLG13par (runs out of memory) compilations is due to the bad handling of conjunctive goals, that results in a exponentially large number of different state transitions. On the other hand, PRP handles conjunctive goals much better in the ABA compilations thanks to the AA progression of the LTL formula. In the *Clerk* problems, the run times of every approach increases exponentially with the problem size. PRP scales slightly worse with the ABA compilation than with the PLG13seq and PLG13par compilations, which can solve 1 and 2 more problems respectively.

Figure 5d compares the size of the policies found by PRP to problems compiled with ABA and PLG13par compilations. PLG13seq compilations resulted in slightly larger policies, due to separate world and sync action phases. We account only for world actions, excluding synchronization actions from the count. Policy sizes with ABA-based compilations are similar, but consistently smaller than those from PLG13par compilations, except in the *Lift* problems where the former results in considerably smaller policies. Finally, we evaluated the validity of our approach with LTL goals that could not be handled by PLG13. In particular, we solved Waldo problems with goals of the form $\Diamond\Box\alpha$.

Collectively, our systems advance the state of the art in LTL-FOND for finite and infinite traces, providing comparable or superior performance to previous LTL-FOND planning methods, while supporting the full expressivity of LTL.

5 Summary and Discussion

We have proposed four compilation-based approaches to FOND planning with LTL goals that are interpreted over either finite or infinite traces. The compilations based on non-deterministic automata collectively capture the full expressivity of LTL for finite and infinite traces and are worst-case exponential in size relative to the LTL. The corresponding LTL-FOND approaches are sound and complete and do not manifest this exponential property in practice, likely due to our encoding optimizations. The approaches based on non-deterministic automata consistently provide superior performance relative to all other approaches. Our two compilations based on alternating automata are linear in size. The corresponding LTL-FOND approaches are sound and incomplete for both the finite and infinite cases. To obtain completeness we would have lost the linear properties of the compilation. Since this restricted but linear compilation was already inferior to our non-deterministic approach, we did not include our complete approach. These compilations address a number of important open problems in FOND planning with LTL, as summarized in Table 1.

Our LTL planning techniques are directly applicable to a number of real-world planning problems as well as capturing a diversity of applications beyond standard planning, including but not limited to genomic rearrangement (Uras and Erdem 2010), story generation (Haslum 2012), business process management (De Giacomo et al. 2014), verification (Albarghouthi, Baier, and McIlraith 2009; Patrizi et al. 2011), and robot motion planning (Plaku 2012).

While LTL is effective for expressing many goals and constraints, some goals are better described procedurally, using regular expressions. The EAGLE goal language (Lago, Pistore, and Traverso 2002; Shaparau, Pistore, and Traverso 2008) and variants of the Golog language (e.g., (Baier, Fritz, and McIlraith 2007)) are prominent among attempts to support planning with action-centric procedural control/goals using regular expressions. Triantafillou, Baier, and McIlraith (2015) combine LTL with regular expression to plan with goals specified in linear dynamic logic (LDL).

Finally, we observe that LTL-FOND is related to LTL synthesis (Pnueli and Rosner 1989). Informally, it is the problem of computing a policy that satisfies an LTL formula, assuming that an adversary may change some fluents after the execution of each action. Recently De Giacomo and Vardi (2015) showed how to map a Finite LTL-FOND problem into a synthesis problem. An open question is whether existing planning technology can be used for LTL synthesis, where the environment is inherently unfair.

Acknowledgements: The authors gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Fondecyt grant numbers 1150328 and 1161526.

References

Albarghouthi, A.; Baier, J. A.; and McIlraith, S. A. 2009. On the use of planning technology for verification. In *Proc. of Validation and Verification of Planning and Scheduling Systems Workshop (VVPS)*.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *AI Magazine* 16:123–191.

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

Baier, J. A.; Fritz, C.; and McIlraith, S. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proc. of the 17th Int'l Conference on Automated Planning and Scheduling (ICAPS)*, 26–33.

Camacho, A.; Triantafillou, E.; Muise, C. J.; Baier, J. A.; and McIlraith, S. A. 2016. Non-deterministic planning with temporally extended goals: Completing the story for finite and infinite LTL (amended version). In *Proceedings of the Workshop on Knowledge-based Techniques for Problem Solving and Reasoning co-located with IJCAI-16*.

Cresswell, S., and Coddington, A. M. 2004. Compilation of LTL goal formulas into PDDL. In *Proc. of the 16th European Conference on Artificial Intelligence (ECAI)*, 985–986.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. of the 23rd Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 854–860.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. In *Proc. of the 24th Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 1558–1564.

De Giacomo, G.; Masellis, R. D.; Grasso, M.; Maggi, F. M.; and Montali, M. 2014. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *Proc. of the 12th Int'l Conference on Business Process Management (BPM)*, volume 8659 of *Lecture notes in Computer Science*, 1–17. Springer.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic-based planner. *AI Magazine* 22(3):95–102.

Edelkamp, S. 2006. Optimal symbolic PDDL3 planning with MIPS-BDD. In *5th Int'l Planning Competition Booklet (IPC-2006)*, 31–33.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

Haslum, P. 2012. Narrative planning: Compilations to classical planning. *Journal of Artificial Intelligence Research* 44:383–395.

Kabanza, F.; Barbeau, M.; and St.-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95(1):67–11.

Kupferman, O., and Rosenberg, A. 2010. The blowup in translating LTL to deterministic automata. In van der Meyden, R., and Smaus, J., eds., *6th International Workshop on Model Checking and Artificial Intelligence (MoChArt)*, volume 6572 of *Lecture Notes in Computer Science*, 85–94. Springer.

Lago, U. D.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proc. of the 18th National Conference on Artificial Intelligence (AAAI)*, 447–454.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-deterministic Planning by Exploiting State Relevance. In *Proc. of the 22th Int'l Conference on Automated Planning and Scheduling (ICAPS)*, 172–180.

Patrizi, F.; Lipovetzky, N.; De Giacomo, G.; and Geffner, H. 2011. Computing infinite plans for LTL goals using a classical planner. In *Proc. of the 22nd Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 2003–2008.

Patrizi, F.; Lipovetzky, N.; and Geffner, H. 2013. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *Proc. of the 23rd Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 2343–2349.

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

Plaku, E. 2012. Planning in discrete and continuous spaces: From LTL tasks to robot motions. In *Advances in Autonomous Robotics - Joint Proceedings of the 13th Annual TAROS Conference and the 15th Annual FIRA RoboWorld Congress*, 331–342.

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 179–190.

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

Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In *Proc. of the 14th European Conference on Artificial Intelligence (ECAI)*, 526–530.

Shaparau, D.; Pistore, M.; and Traverso, P. 2008. Fusing procedural and declarative planning goals for nondeterministic domains. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, 983–990.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *Proc. of the 24th Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 1696–1703.

Triantafillou, E.; Baier, J.; and McIlraith, S. A. 2015. A unifying framework for planning with LTL and regular expressions. In *Workshop on Model-Checking and Automated Planning (MOCHAP) at ICAPS*.

Uras, T., and Erdem, E. 2010. Genome rearrangement: A planning approach. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, 1963–1964.

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

Vardi, M. Y. 1995. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, volume 1043 of *Lecture notes in Computer Science*, 238–266. Springer.