Non-Deterministic Planning with Temporally Extended Goals: Completing the story for finite and infinite LTL

Alberto Camacho[†], Eleni Triantafillou[†], Christian Muise[*], Jorge A. Baier[‡], Sheila A. McIlraith[†]
[†]Department of Computer Science, University of Toronto
[*]CSAIL, Massachusetts Institute of Technology
[†]Departamento de Ciencia de la Computación, 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 planning with temporally extended goals over both finite and infinite traces where actions can be non-deterministic, and where temporally extended goals are specified in linear temporal logic (LTL). Unlike existing LTL planners, we place no restrictions on our LTL formulae beyond those necessary to distinguish finite from infinite trace interpretations. We realize our planner by compiling temporally extended goals, represented in LTL, into Planning Domain Definition Language problem instances, and exploiting a state-of-the-art fully observable non-deterministic planner to compute solutions. The resulting planner is sound and complete. Our approach exploits the correspondence between LTL and automata. We propose several different compilations based on translations of LTL to (Büchi) alternating or non-deterministic finite state 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. We do so while demonstrating competitive performance relative to the state of the art in LTL planning.

1 Introduction

Most real-world planning problems involve complex goals that are temporally extended, require adherence to safety constraints and directives, necessitate the optimization of preferences or other quality measures, and/or require or may benefit from following a prescribed high-level script that specifies *how* the task is to be realized. In this paper we focus on the problem of planning for temporally extended goals, constraints, directives or scripts that are expressed in Linear Temporal Logic (LTL) for planning domains in which actions can have *non-deterministic* effects, and where LTL is interpreted over either finite or infinite traces.

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 exists. Unfortunately, the latter approach is limited to the proper subset of LTL for which there exists a *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.

In this paper, we propose a number of compilation-based approaches for LTL planning with non-deterministic actions. Specifically, we present two approaches for LTL planning with non-deterministic actions over infinite traces and two approaches for LTL planning with non-deterministic actions over finite traces[1]. In each case, we exploit translations from LTL to (Büchi) alternating or non-deterministic finite state automata. All of our compilations are sound and complete and result in Planning Domain Definition Language (PDDL) encodings suitable for input to standard fully observable non-deterministic (FOND) planners. Our compilations based on alternating automata are linear in time and space with respect to the size of the LTL formula, while those based on non-deterministic finite state automata are worst-case exponential in time and space (although optimizations in the implementation avoid this in our experimental analysis).

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 reasonably straightforward, the infinite case required non-trivial insights and modifications to Torres and Baier's approach. We evaluate the relative performance of our compilation-based approaches using state-of-the-art FOND planner PRP (Muise, McIlraith, and Beck 2012), demonstrating that they are competitive with state-of-the-art LTL planning techniques.

[1]Subtleties relating to the interpretation of LTL over finite traces are discussed in (De Giacomo and Vardi 2013).

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 complete spectrum of FOND planning with LTL goals interpreted over infinite traces. Indeed, all of our translations capture the full expressivity of the LTL language. Table 1 summarizes existing compilation-based approaches and the contributions of this work. Our compilations enable a diversity of real-world planning problems as well as supporting a number of applications outside planning proper ranging from business process analysis, and web service composition to narrative generation, automated diagnosis, and automated verification. Finally and 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 consists of a tuple $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$, where $\mathcal{F}$ is a set of propositions that we call *fluents*, $\mathcal{I} \subseteq \mathcal{F}$ characterizes what holds in the initial state; $\mathcal{G} \subseteq \mathcal{F}$ characterizes what must hold for the goal to be achieved. Finally $\mathcal{A}$ 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, 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 some 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 the planning 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. Together these deterministic actions comprise the set $\mathcal{A}'$.

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. The *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 by means of *finite-state controllers* (FSCs). Formally, a FSC is a tuple $\Pi = \langle C, c_0, \Gamma, \Lambda, \rho, \Omega \rangle$, where C is the set of *controller states*, $c_0 \in C$ is the *initial controller state*, $\Gamma = S$ is the *input alphabet* of Π, $\Lambda = \mathcal{A}$ is the *output alphabet* of Π, $\rho : C \times \Gamma \rightarrow C$ is the *transition function*, and $\Omega : C \rightarrow \Lambda$ is the controller *output function* (cf. (Geffner and Bonet 2013; Patrizi, Lipovetzky, and Geffner 2013)). In a planning state s, Π outputs action $\Omega(c_i)$ when the controller state is c_i. Then, the controller transitions to state $c_{i+1} = \rho(c_i, s')$ if s' is the new planning state, assumed to be fully observable, that results from applying $\Omega(c_i)$ in s. The *execution* of a FSC Π in controller state c (assumed to be $c = c_0$) and 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 such that all of its state-action-state substrings s_i, a_i, s_{i+1} are such that $\Omega(c_i) = a_i$, s_{i+1} is a result of applying a_i in s_i, and $c_{i+1} = \rho(c_i, s_i)$. Finite executions ending in a state s_n are such that $\Omega(c_n)$ is undefined. An execution σ *yields* the state trace π that results from removing all the action symbols from σ.

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 is a *strong cyclic plan* for $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ 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.

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. Because the execution of a sequence of actions induces a trace of planning states, LTL can be naturally used to specify temporally extended planning goals when the execution of the plan naturally yields an infinite state trace, as may be the case in non-deterministic planning.

In classical planning –i.e. planning with deterministic actions and final-state goals–, plans are finite sequences of actions which yield finite execution traces. As such, approaches to planning with deterministic actions and LTL goals (e.g., (Baier and McIlraith 2006)), including the Planning Domain Definition Language (PDDL) version 3 (Gerevini and Long 2005), use a *finite* semantics for LTL, whereby the goal formula is evaluated over a finite state trace. De Giacomo and Vardi (2013) formally described and analyzed such a version of LTL, which they called $\mathsf{LTL_f}$, noting the distinction with LTL (De Giacomo, Masellis, and Montali 2014).

LTL and $\mathsf{LTL_f}$ allow the use of modal operators *next* ($\bigcirc$), and *until* (U), from which it is possible to define the well-known operators *always* ($\Box$) and *eventually* ($\Diamond$). $\mathsf{LTL_f}$, in addition, allows a *weak next* ($\bullet$) operator. An $\mathsf{LTL_f}$ 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 $\bullet\psi$. LTL is defined analogously.

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

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) (EXP) **[this paper (BAA)]** (LIN) **[this paper (NBA)]** (EXP)	[Edelkamp, 2006] (EXP) [Cresswell & Coddington, 2006] (EXP) [Baier & McIlraith, 2006] (EXP) [Torres & Baier, 2015] (LIN)	**[this paper (NFA)]** (EXP) **[this paper (AA)]** (LIN)

Table 1: Automata-based compilation approaches for LTL planning. (EXP): worst case exponential. (LIN): linear.

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 natural number $i \geq 1$:

- $\pi, i \models p$, 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. Therefore, henceforth 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 $(\mathsf{true}\,\mathsf{U}\,\varphi)$, and $\Box\varphi$ as $\neg\Diamond\neg\varphi$. We use the *release* operator, defined by $(\psi\,\mathsf{R}\,\chi) \stackrel{\text{def}}{=} \neg(\neg\psi\,\mathsf{U}\,\neg\chi)$.

2.3 LTL, Automata, and Planning

Regardless of whether the interpretation is over an infinite or finite trace, given an LTL formula φ there exists an automata $\mathcal{A}_\varphi$ that accepts a trace π iff $\pi \models \varphi$. For infinite interpretations of φ, a trace π is accepting when the run of (a Büchi non-deterministic automata) $\mathcal{A}_\varphi$ on π visits accepting states infinitely often. For finite interpretations, π is accepting when the final automata state is accepting. For the infinite case such automata may be either Büchi non-deterministic or Büchi alternating (Vardi and Wolper 1994), whereas for the finite case such automata may be either non deterministic (Baier and McIlraith 2006) or alternating (De Giacomo, Masellis, and Montali 2014; Torres and Baier 2015). Alternation allows the generation of compact automata; specifically, $\mathcal{A}_\varphi$ is linear in the size of φ (both in the infinite and finite case), whereas the size of non-deterministic (Büchi) automata is worst-case exponential.

These automata constructions have been exploited in deterministic and non-deterministic planning with LTL via compilation approaches that allow us to use existing planning technology for non-temporal goals. The different state of the art automata-based approaches for deterministic and FOND LTL planning are summarized in Table 1. Patrizi, Lipovetzky, and Geffner (2013) present a Büchi automata-based compilation for that subset of LTL which relies on the construction of a Büchi *deterministic* automata. It is a well-known fact that Büchi deterministic automata are not equivalent to Büchi non-deterministic automata, and thus this last approach is applicable to a limited subset of LTL formulae.

3 FOND Planning with LTL Goals

An LTL-FOND planning 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 problem are FSCs, as described below.

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* $\pi \models \varphi$*.*

Below we present two general approaches to solving LTL-FOND planning problems by compiling them into standard FOND problems. Each exploits correspondences between LTL and either alternating or non-deterministic automata, and each is specialized, as necessary, to deal with LTL interpreted over either infinite (Section 3.1) or finite (Section 3.2) traces. We show that FSC representations of strong-cyclic solutions to the resultant FOND problem are solutions to the original LTL-FOND problem. Our approaches are the first to address the full spectrum of FOND planning with LTL interpreted over finite and inifinte 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 approaches to infinite LTL-FOND planning. The first approach exploits Büchi alternating automata (BAA) and is linear in time and space with respect to the size of the LTL formula. The second approach exploits Büchi non-deterministic automata (NBA), and is worst-case exponential in time and space with respect to the size of the LTL formula. Nevertheless, as we see in Section 4, the second compilation does not exhibit this worst-case complexity in practice, generating high quality solutions with reduced compilation run times and competitive search performance.

3.1.1 A BAA-based Compilation Our BAA-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 translation for the infinite case, and thus we needed to modify it significantly. This is because the accepting condition for BAAs is more involved than that of regular AAs.

The first step in the compilation is to build a BAA 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 BAA we use below is an adaptation of the BAA by Vardi (1995). Formally, it is represented by a tuple $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_\varphi, Q_{Fin})$, where the set of states, Q, is the set of subformulae of φ, $sub(\varphi)$ (including φ), Σ contains all sets of propositions in $\mathcal{P}$, $Q_{Fin} = \{\alpha \,\mathsf{R}\, \beta \in sub(\varphi)\}$, and the transition function, δ is given by:

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

As a note for the reader unfamiliar with BAAs, the transition function for these automata takes a state and a symbol and returns a positive Boolean formula over the set of states Q. Furthermore, a *run* of a BAA over an infinite string $\pi = s_1 s_2 \ldots$ 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 a BAA 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 BAA 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.

In our BAA translation for LTL-FOND we follow a similar approach to that developed in the TB15 translation: given an input problem $\mathcal{P}$, we generate an equivalent problem $\mathcal{P}'$ in which we represent the configuration of the BAA with fluents (one fluent q per each state q of the BAA). $\mathcal{P}'$ contains the actions in $\mathcal{P}$ plus additional *synchronization actions* whose objective is to update the configuration of the BAA. In $\mathcal{P}'$, there are special fluents to 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 BAA is updated.

Before providing details of the translation we overview the main differences between our translation 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 only contains final BAA states. Thus, when building a plan with our translation, the planner is given the ability to "decide" 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*.

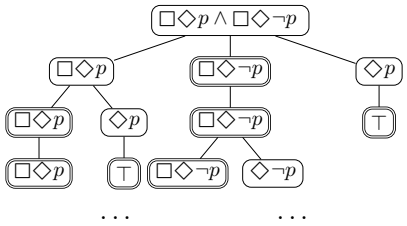


Figure 1: An accepting run of a BAA for $\Box\Diamond p \wedge \Box\Diamond\neg p$ over an infinite sequence of states in which the truth value of p alternates. Double-line ovals are accepting states/conditions.

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 BAA $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_\varphi, Q_{Fin})$, the translated FOND problem is $\mathcal{P}' = \langle \mathcal{F}', \mathcal{I}', \mathcal{G}', \mathcal{A}' \rangle$, where each component is described below.

Fluents $\mathcal{P}'$ has the same fluents as $\mathcal{P}$ plus fluents for the representation of the states of the automaton $F_Q = \{q_\psi \mid \psi \in Q\}$, and flags $\mathbf{copy}$, $\mathbf{sync}$, $\mathbf{world}$ for controlling the different modes. Finally, it includes the set $F^S_Q = \{q^S_\psi \mid \psi \in Q\}$ which are *copies* of the automata fluents, and *tokens* $F^T_Q = \{q^T_\psi \mid \psi \in Q\}$. 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 $\mathcal{A}_w$:

$$Pre_{a'} = Pre_a \cup \{\mathbf{world}\},$$
$$Eff_{a'} = Eff_a \cup \{\mathbf{copy}, \neg\mathbf{world}\}.$$

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:

$$Eff_{copy} = \{q \rightarrow \{q^S, \neg q\} \mid q \in F_Q\} \cup \{\mathbf{sync}, \neg\mathbf{copy}\}$$

As soon as the **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 2. These actions update the state of the automaton following the definition of the transition function, δ. In addition, each synchronization action for a formula ψ that has 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 actions are executable: $world$ and $continue$. The objective of $world$ action is to reestablish world mode. Its precondition is $\{\mathbf{sync}\} \cup \overline{F^S_Q}$, and its effect is $\{\mathbf{world}, \neg\mathbf{sync}\}$.

The $continue$ action also reestablishes world mode, but in addition "decides" that an accepting BAA can be reached in the future. This is reflected by the non-deterministic effect that makes the fluent **goal** true. As such, it "tokenizes" all states that are not final states in F_Q, by adding q^T for each BAA state q that is non-final and currently true. Formally,

$$Pre_{continue} = \{\mathbf{sync}\} \cup \{\neg q^T_\varphi \mid \varphi \notin Q_{Fin}\}$$
$$Eff_{continue} = \{\{\mathbf{goal}\},$$
$$\{q_\varphi \rightarrow q^T_\varphi \mid \varphi \notin Q_{Fin}\} \cup \{\mathbf{world}, \neg\mathbf{sync}\}\}$$

The set $\mathcal{A}_s$ is defined as the one containing 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 $\mathcal{G}' = \{\mathbf{goal}\}$.

In summary, our BAA-based approach builds on TB15 while integrating ideas from PLG13. Like PLG13 our approach uses a $continue$ action to find plans with lassos, but unlike PLG13, our translation 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^T_Q, which did not exist in TB15, are created for each non-accepting state and can only be eliminated when a non-accepting BAA state becomes accepting.

Now we show how, given a strong cyclic policy for $\mathcal{P}'$, we can generate an FSC for $\mathcal{P}$. Observe that every state ξ, which is a set of fluents in F', can be written as the disjoint union of sets $s_w = \xi \cap F$ and $s_q = \xi \cap (F' \setminus F)$. Abusing notation, we use $s_w \in 2^F$ to represent a state in $\mathcal{P}$. For a planning state $\xi = s_w \cup s_q$ green in which $p(\xi)$ is defined, we define $\Omega(\xi)$ to be the action in $\mathcal{A}$ whose translation is $p(\xi)$. Recall now that executions of a strong-cyclic policy p for $\mathcal{P}'$ in state ξ generate plans of the form $a_1\alpha_1a_2\alpha_2\ldots$ where each a_i is a world action in $\mathcal{A}_w$ and α_i are sequences of actions in $\mathcal{A}' \setminus \mathcal{A}_w$. Thus $\Omega(\xi)$ can be generated by taking out the fluents **world** and **copy** from the precondition and effects of $p(\xi)$. If state s'_w is a result of applying $\Omega(\xi)$ in s_w, we define $\rho(\xi, s'_w)$ to be the state ξ' that results from the composition of consecutive non-world actions α_1 mandated by an execution of p in $s'_w \cup s_q$. Despite non-determinism in the executions, the state $\xi' = \rho(\xi, s'_w)$ is well-defined.

The BAA translation for LTL-FOND is sound and complete. Throughout the paper, the soundness property guarantees that FSCs obtained from solutions to the compiled problem $\mathcal{P}'$ are solutions to the LTL-FOND problem $\mathcal{P}$, whereas the completeness property guarantees that a solution to $\mathcal{P}'$ exists if one exists for $\mathcal{P}$.

Theorem 1. *The BAA translation for Infinite* LTL*-FOND planning is sound, complete, and linear in the size of the goal formula.*

A complete proof is not included but we present some of the intuitions our proof builds on. Consider a policy p' for $\mathcal{P}'$. p' yields three types of executions: (1) finite executions that end in a state where **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 a BAA accepting state turns into an accepting state. This in turn means that every branch of the run contains an infinite repetition of final states. The plan for $\mathcal{P}$, p, is obtained by removing all synchronization actions from p', and the FSC that is solution to $\mathcal{P}$ is obtained as described above. In the other direction, a plan p' for $\mathcal{P}'$ can be built from a plan p for $\mathcal{P}$ by adding synchronization actions. Theorem 1 follows from the argument given above and reuses most of the argument that TB15 uses to show their translation is correct.

3.1.2 An NBA-based Compilation This compilation relies on the construction of a non-determinisitic Büchi automaton (NBA) for the goal formula, and builds on translation 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 translation 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 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 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 translation 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 original non-deterministic actions from their determinized counterparts. More precisely, the translation of a non-deterministic action a in $\mathcal{P}$ is a non-deterministic action a' in $\mathcal{P}'$ that is constructed by first determinizing a into a set of actions, a_i that correspond to each of the non-deterministic outcomes of a, applying the BM06-based translation to each a_i to produce a'_i, and then reassembling the a'_is back into a non-deterministic action, a'. In so doing, $Eff_{a'}$ is the set of outcomes in each of the deterministic actions a'_i, and $Pre_{a'}$ is similarly the precondition of any of these a'_i.

The modification of the BM06 translation used in the second phase leverages ideas present in PLG13 and our BAA-based compilations to capture infinite runs via induced non-determinism. In particular, it includes a $continue$ action whose precondition is the accepting configuration of the NBA (a disjunction of the fluents representing accepting states). Unlike our BAA-based compilation, the tokenization is not required because accepting runs are those that achieve accepting states infinitely often, no matter which ones. As before, one non-deterministic effect of $continue$ is to achieve **goal**, while the other is to force the planner to perform at least one action. This is ensured by adding an extra precondition to $continue$, **can_continue**, which is true in the initial state, it is made true by every action but $continue$, and is deleted by $continue$.

In order to construct a solution Π to $\mathcal{P}$ from a strong-cyclic solution p to $\mathcal{P}' = \langle \mathcal{F}', \mathcal{I}', \mathcal{G}', \mathcal{A}' \rangle$, it is useful to represent states ξ in $\mathcal{P}'$ as the disjoint union of $s = \xi \cap F$ and $q = \xi \cap (F' \setminus F)$. Intuitively, s represents the planning state in $\mathcal{P}$, and q represents the automaton state. The controller $\Pi = \langle C, c_0, \Gamma, \Lambda, \rho, \Omega \rangle$ is defined as follows. $c_0 = I'$ is the initial controller state; $\Gamma = 2^{\mathcal{F}}$; $\Lambda = \mathcal{A}$; $\rho(\xi, s') = s' \cup q'$, where q' is the automaton state that results from applying action $p(\xi)$ in ξ; $\Omega(\xi) = p(\xi)$; and $C \subseteq 2^{\mathcal{F}'}$ is the domain of p. Actions in $\mathcal{P}'$ are non-deterministic and have conditional effects, but the automaton state q' that results from applying action $p(\xi)$ in state $\xi = s \cup q$ is deterministic, and thus ρ is well-defined.

Theorem 2. *The NBA translation for* Infinite LTL*-FOND planning is sound, complete, and worst-case exponential in the size of the* LTL *formula.*

Theorem 2 follows from soundness, completeness, and the complexity of the BM06 translation, this time using a NBA automaton, and an argument similar to that of Theorem 1. This time, if *continue* executes infinitely often we can guarantee accepting NBA states are reached infinitely often.

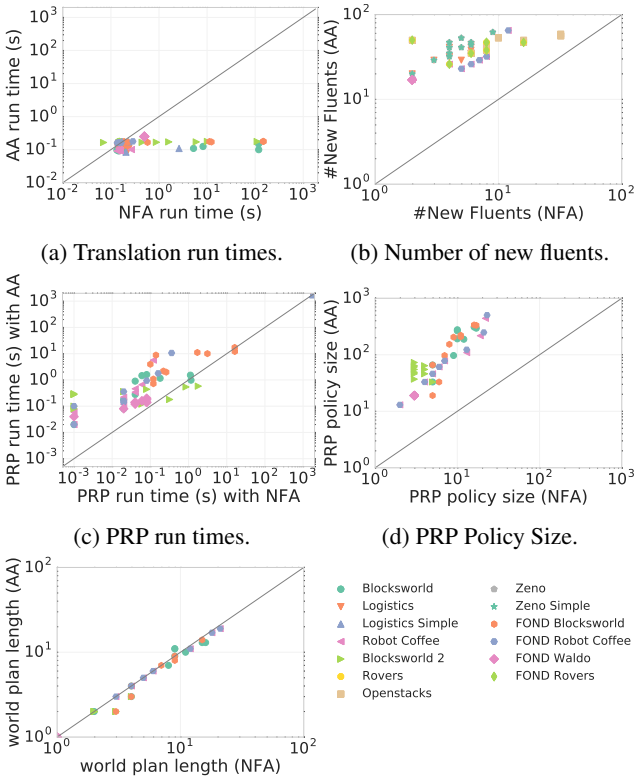
3.2 From Finite LTL-FOND to FOND

Our approach to finite LTL-FOND extends the BM06 and TB15 translations, originally intended for *finite* LTL planning with *deterministic* actions, to the non-deterministic action setting. Both the original BM06 and TB15 translations share two general steps. In step one, the LTL goal formula is translated to 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 the automaton. In the case of BM06 these capture the state of the automaton and how domain actions cause the state of the automaton to be updated. In the case of the TB15 translation, $\mathcal{P}$ must also be augmented with synchronization actions. Finally, in both cases the original problem goals must be modified to capture the accepting states of automata.

When BM06 and TB15 are exploited for LTL-FOND, the non-deterministic nature of the actions must be taken into account. This is done in much the same as with the NBA- and BAA-based compilations described in the previous section. In particular, non-deterministic actions in the LTL-FOND problem are determinized, the BM06 (resp. TB15) translation is applied to these determinized actions, and then the non-deterministic actions reconstructed from their translated determinized counterparts (as done in the NBA-based compilation) to produce FOND problem, $\mathcal{P}'$. A FSC solution, Π, to the LTL-FOND problem $\mathcal{P}$, can be obtained from a solution to $\mathcal{P}'$. When the NFA-based translations are used, the FSC, Π, is obtained from policy p following the approach described for NBA-based translations. When the AA-based translations are used, the FSC, Π, is obtained from p following the approach described for BAA-based translations.

Theorem 3. *The NFA (resp. AA) translation for Finite* LTL*-FOND is sound, complete, and exponential (resp. linear) in the size of the* LTL *formula.*

Soundness and completeness in Theorem 3 follows from soundness and completeness of the BM06 and TB15 translations. Fair executions of Π yield finite plans for $\mathcal{P}'$, and therefore state traces (excluding intermediate synchronization states) satisfy φ. Conversely, our approach is complete as for every plan in $\mathcal{P}$, one can construct a plan in $\mathcal{P}'$. Finally, the run-time complexity and size of the translations is that of the original BM06 and TB15 translations – worst case exponential in time and space for the NFA-based approach and linear in time and space for the AA approach.

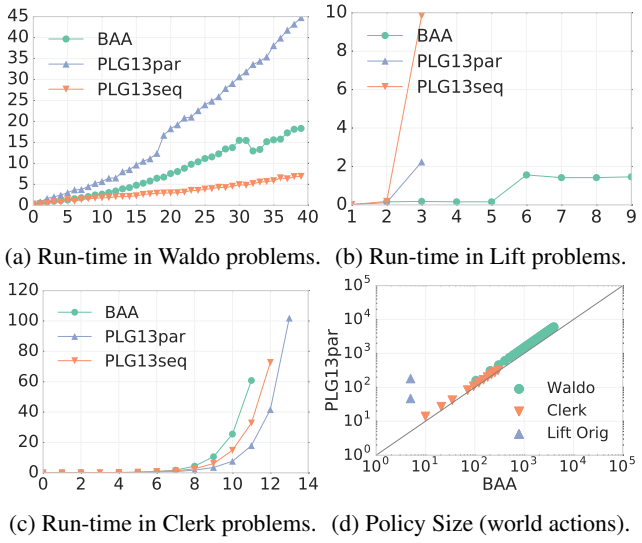


(a) Translation run times. (b) Number of new fluents.

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

(e) Preferred *world* Plan Length.

Figure 2: Performance of our planning system using AA- and NFA-based translations in different problems with deterministic and non-deterministic actions and *finite* LTL goals.



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

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

Figure 3: Performance of our planning system using BAA-based translations in different LTL-FOND domains. We report PRP run-times (in seconds) and policy sizes, excluding synchronization actions.

4 Experiments

We evaluate our framework on a selection of benchmark domains with LTL goals from (Baier and McIlraith 2006; Patrizi, Lipovetzky, and Geffner 2013; Torres and Baier 2015), modified to include non-deterministic actions. Experiments were conducted on an Intel Xeon E5-2430 CPU @2.2GHz Linux server, using a 4GB memory and a 30-minute time limit.

LTL-FOND Planning over Finite Traces: We evaluated the performance of our BM06 (NFA) and TB15 (AA) translators, with respect to a collection of problems with deterministic and non-determinisitic actions and LTL goals, interpreted on finite traces. We used the state-of-the-art FOND planner, PRP (Muise, McIlraith, and Beck 2012), to solve the translated problems. NFA-based translation times increased when the LTL formula had a large number of conjunctions and nested modal operators, whereas AA-based translation times remain negligible. However, the AA translation included a number of new fluents that were, in some cases, up to one order of magnitude larger than with the NFA (Figures 2a and 2b). This seems to translate into more complex problems, as PRP run times become almost consistently greater in problems translated with AA (Figure 2c). The size of the policies obtained from the AA compilations were considerably greater than those obtained with NFA compilations (Figure 3d). This is expected, as AA translations introduce a number of synchronization actions, whereas the number of actions in NFA translations remains unchanged. To assess the quality of the plans obtained from each translation, we compared the number of *world* actions (i.e., excluding automaton-state *synchronization* actions) in the shortest plans of the policies obtained (Figure 2e). This is a crude estimator of the quality of plans, since these plans are not necessarily the ones that minimize the number of world actions, as they also contain synchronization actions. The number of world actions that we obtained in both compilations was very similar.

Interestingly, whereas the size of the AA translations is linear in the size of the original LTL formula and NFA translations are worst-case exponential, in practice we observed the size of the NFA-based translated problems is smaller. Furthermore, PRP performs better when problems are compiled using NFAs, generating similar quality policies in lower search run-times.

We didn't experience any untoward decrease in performance in deterministic problems that were extended with non-deterministic actions, suggesting that AA- and NFA-based translations remain competitive in LTL-FOND.

LTL-FOND Planning over Infinite Traces: The relative performance observed between NBA- and BAA-based translations for LTL-FOND planning, interpreted over infinite traces, is reflective of the finite case. NBA translation run times are greater, but result in lower planner run times and smaller policy sizes. For reference, we compared BAA translations with the so-called *sequential* and *parallel* translations developed by Patrizi, Lipovetzky, and Geffner (2013), subsequently referrd 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 translations forced us to perform such comparisons only in 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. Unfortunately, we could not include a fair comparison with NBA translations in the *Lift* and *Clerk* domains, due to a specific encoding that forced transitions to synchronization phases (existing in PLG13 and BAA translations, but not in NBA). In the *Waldo* problems, however, NBA translations generated smaller solutions (by a half) with roughly half the run time required by BAA. On the other hand, NBA translation times timed out after the twelfth instance (possibly due to an unoptimized implementation of the translator).

The *Waldo* problems require construction of a controller for a robot that moves around n rooms and finds Waldo infinitely often. Waldo may or may not appear in the n-th and $n/2$-th rooms when these are visited. The dynamics of the problem preclude visiting a room twice before visiting the remaining ones, in which case the predicate $search\_again$ becomes true. The LTL goal of the problem is $\Box\Diamond search\_again \vee Waldo$. The *Lift* problems requires construction of a controller for an n-floor building that serves all requests. The dynamics of the problem require alternation between $move$ and $push\_f_i$ actions, $i = 1, \ldots, n$. Fluents at_i and req_i model, respectively, whether the lift is at the i-th floor, and whether a request from the i-th floor has been issued and not served. The lift can only move up if some request is issued. The $push\_f_i$ actions nondeterministically request the lift to service the i-th floor. Initially, the lift is at floor 1, and no request is issued. The LTL goal of the problem is $\varphi = \bigwedge_{i=1}^{n} \Box\Diamond(req_i \rightarrow at_i)$. Finally, the *Clerk* problems require construction of a controller that serves all clients in a store. Clients can order one of n packages p_i. If the package is not available, the clerk has to buy it from a supplier, pick it up, and store it in its correct location. In order to serve the client, the clerk has to find the package, pick it up, and sell it. The LTL goal of the problem is $\Box(active\_request \rightarrow \Diamond(item\_served \vee item\_stored))$.

The results of experiments are summarized in Figure 3. In *Waldo* problems, the planner run times using BAA-based translations are situated between the run times with PLG13seq and PLG13par. In *Lift* problems, the BAA translations 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) translations is due to the bad handling of conjunctive goals, that results in a exponentially large number of different state transitions. On the other hand, the PRP handles conjunctive goals much better in the BAA translations thanks to the AA progression of the LTL formula. In the *Clerk* problems, PRP scales slightly worse with the BAA translation than with the PLG13seq and PLG13par translations, which can solve 1 and 2 more problems respectively. The run times with all translations seem to show the same exponential trend, and differ in a small offset that corresponds to the increase in problem complexity.

Figure 3d compares the size of the policies found by PRP to problems compiled with BAA and PLG13par translations. PLG13seq translations 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 BAA-based translations are similar, but consistently smaller than those from PLG13par translations, except in the *Lift* problems where the former results in considerably smaller policies. Finally, we evaluated the validity of our system with LTL goals that could not be handled by PLG13. In particular, we solved Waldo problems with goals of the form $\Diamond\Box\alpha$.

Overall, our system proves very competitive with (as good as or better than) the previous state-of-the-art LTL-FOND planning methods, while supporting a much broader spectrum (the full spectrum) of LTL formulae.

5 Summary and Discussion

We have proposed four compilation-based approaches to fully observable non-deterministic planning with LTL goals that are interpreted over either finite or infinite traces. These compilations support the full expressivity of LTL, in contrast to much existing work. In doing so, we address a number of open problems in planning with LTL with non-deterministic actions, as noted in Table 1. Our LTL planning techniques are directly applicable to a number of real-world planning problems that are not captured by existing systems. Furthermore they are useful in a diversity of applications beyond standard planning, including but not limited to genomic rearrangement (Uras and Erdem 2010), program test generation (Razavi, Farzan, and McIlraith 2014), story generation (Haslum 2012), automated diagnosis (Grastien et al. 2007; Sohrabi, Baier, and McIlraith 2010), business process management (De Giacomo et al. 2014) and verification (Albarghouthi, Baier, and McIlraith 2009; Patrizi et al. 2011).

We evaluated the effectiveness of our FOND compilations using the state-of-the-art FOND planner, PRP. An interesting observation is that our worst-case exponential NFA-based translations run faster and return smaller policies than the AA-based linear translations. This seems to be due to the larger number of fluents (and actions) required in the AA-based translations. Compared to the existing approach of (Patrizi, Lipovetzky, and Geffner 2013), experiments indicate that our approaches scale up better.

Finally, we observe that LTL-FOND is related to the problem of LTL synthesis (Pnueli and Rosner 1989). Informally, it is the problem of computing a policy that satisfies an LTL formula, assuming that an adversary (which we can associate to the non-deterministic environment) 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. Sardiña and D'Ippolito (2015) go further, showing how FOND plans can be synthesized using LTL synthesis algorithms. An open question is whether any existing planning technology can be used for LTL synthesis as well. LTL synthesis is not an instance of strong cyclic FOND planning since synthesis adversaries are not fair.

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

References

Albarghouthi, A.; Baier, J. A.; and McIlraith, S. A. 2009. On the use of planning technology for verification. In *Proceedings of the 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 *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 788–795.

Cresswell, S., and Coddington, A. M. 2004. Compilation of LTL goal formulas into PDDL. In *Proceedings 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 *Proceedings of the 23rd International 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 *Proceedings of the 24th International 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 *Proceedings of the 12th International Conference on Business Process Management (BPM)*, volume 8659 of *Lecture notes in Computer Science*, 1–17. Springer.

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 (AAAI)*, 1027–1033.

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 International 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.

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*, 305–310.

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.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-deterministic Planning by Exploiting State Relevance. In *Proceedings of the 22th International 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 *Proceedings of the 22nd International 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 *Proceedings of the 23rd International 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 *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 479–484.

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 *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, 46–57.

Razavi, N.; Farzan, A.; and McIlraith, S. A. 2014. Generating effective tests for concurrent programs via AI automated planning techniques. *International Journal on Software Tools for Technology Transfer* 16(1):49–65.

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

Sardiña, S., and D'Ippolito, N. 2015. Towards fully observable non-deterministic planning as assumption-based automatic synthesis. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 3200–3206.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2010. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 26–36.

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

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

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.