Fair LTL Synthesis for Non-Deterministic Systems using Strong Cyclic Planners

Fabio Patrizi
Sapienza University of Rome
Rome, Italy
patrizi@dis.uniroma1.it

Nir Lipovetzky
The University of Melbourne
Melbourne, Australia
nir.lipovetzky@unimelb.edu.au

Hector Geffner
ICREA & Univ. Pompeu Fabra
Barcelona, Spain
hector.geffner@upf.edu

Abstract

We consider the problem of planning in environments where the state is fully observable, actions have non-deterministic effects, and plans must generate infinite state trajectories for achieving a large class of LTL goals. More formally, we focus on the control synthesis problem under the assumption that the LTL formula to be realized can be mapped into a *deterministic* Büchi automaton. We show that by assuming that action non-determinism is *fair*, namely that infinite executions of a non-deterministic action in the same state yield each possible successor state an infinite number of times, the *(fair) synthesis problem* can be reduced to a *standard strong cyclic planning task over reachability goals*. Since strong cyclic planners are built on top of efficient classical planners, the transformation reduces the non-deterministic, fully observable, temporally extended planning task into the solution of classical planning problems. A number of experiments are reported showing the potential benefits of this approach to synthesis in comparison with state-of-the-art symbolic methods.

1 Introduction

Classical planning is concerned with reachability problems over compact propositional representations where a *goal state* is to be achieved from a given initial state by applying actions with deterministic effects. While the problem is computationally intractable [Bylander, 1994], significant progress has been achieved, enabling classical planners to solve problems with hundreds of actions and propositions [Richter and Westphal, 2010]. In the last few years *temporally extended goals*, expressed in temporal logics such as LTL [Pnueli, 1977] have been increasingly used to capture a richer class of plans, where restrictions over the whole sequence of states must be satisfied as well [Gerevini and Long, 2005]. A (temporally) extended goal may state, for example, that any borrowed tool should be kept clean until returning it; a constraint that does not apply to states but, rather, to state sequences. While most of this work has been focused on LTL goals that can be achieved by finite plans [Bacchus and Kabanza, 1998; Edelkamp, 2003; Cresswell and Coddington, 2004; Edelkamp, 2006; Baier and McIlraith, 2006; Baier *et al.*, 2009], more recent work has addressed the more general task of planning for LTL goals that require infinite plans [Kabanza and Thiébaux, 2005; Albarghouthi *et al.*, 2009; Patrizi *et al.*, 2011]. For instance, in order to monitor a set of rooms, an extended LTL goal may require the agent to always return to each of the rooms, a goal that cannot be achieved by a finite plan. It is known that the required plans in such a setting can be finitely characterized as "lassos": a sequence of actions mapping the initial state of a composite system into some state s, followed by a second action sequence that maps s into itself that is repeated infinitely often [Vardi, 1996]. In [Patrizi *et al.*, 2011], it is shown that such plans can efficiently be constructed by calling a *classical planner* once, over a classical planning problem obtained from the composite system represented by the product of the planning domain and the Büchi automaton corresponding to the LTL goal [De Giacomo and Vardi, 1999].

The aim of this work is to push the envelope further by showing how to use current planning techniques for dealing with the more general problem of planning for LTL goals that require *infinite executions,* in environments where actions have *non-deterministic* effects and states are *fully observable*. From a planning point of view, this is the standard fully-observable non-deterministic planning problem (FOND) but with reachability goals replaced by temporally extended goals. From a formal verification point of view, this is the standard control synthesis problem for LTL formulas [Pnueli and Rosner, 1989]. Since the general synthesis problem is known to be 2EXPTIME-complete [Pnueli and Rosner, 1989], however, it is common to impose restrictions on the class of LTL formulas for scaling up. A common restriction is to focus on Generalized Reactive or GR(1) formulas for which a synthesis algorithm has been developed that is $O(|S|^3)$, where S is the problem state space [Bloem *et al.*, 2012]. Other approaches are based on restricting to LTL formulas that can be mapped efficiently into *deterministic automata* [Alur and La Torre, 2004; Morgenstern and Schneider, 2008; 2011].

In this work we adopt the latter assumption by considering LTL formulas that can be mapped into deterministic Büchi automata, and show that by assuming further that action non-determinism is *fair*, namely that infinite executions of a non-deterministic action in the same state yield

each possible successor state an infinite number of times, the *(fair) synthesis problem* can be reduced to a *standard strong cyclic planning task* over a *domain* that is the product of the (non-deterministic) planning domain and the (deterministic) Büchi automaton. A strong cyclic plan over a non-deterministic planning domain is a policy π such that if s is a non-goal state that is potentially reachable from the initial state by following π, then a goal state must be potentially reachable from s by following π [Daniele *et al.*, 1999; Cimatti *et al.*, 2003]. Since strong cyclic planners are built on top of efficient classical planners [Kuter *et al.*, 2008; Fu *et al.*, 2011; Muise *et al.*, 2012] the transformation reduces the non-deterministic, fully observable, extended planning task into the solution of classical planning tasks. Finally, we report on a number of experiments showing the potential benefits of this planning approach to synthesis in comparison with state-of-the-art symbolic synthesis methods.

The paper is organized as follows. First, we review nondeterministic planning domains, LTL, and Büchi automata. Then we consider the target problem and the reduction to strong cyclic planning, present the empirical results, and finish with a summary and discussion.

2 Preliminaries

We review the models and results associated with nondeterministic planning, LTL, and Büchi automata.

2.1 Nondeterministic Planning Domains

A *(nondeterministic) planning domain* is a tuple $\mathcal{D} = \langle Act, Prop, S, S_0, f\rangle$ where:

- Act is the finite set of domain actions;
- $Prop$ is the set of domain propositions;
- $S \subseteq 2^{Prop}$ is the set of domain states;
- $s_0 \in S$ is the (single) initial state; and
- $f : S \times Act \mapsto 2^S$ is the state-transition function.

A $\mathcal{D}$*-path* is a possibly infinite sequence $\mu = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ such that $s_{i+1} \in f(s_i, a_i)$. Astate $s \in S$ is *reachable* if there exists a $\mathcal{D}$-path where s occurs. Moreover, we say that a path μ *reaches* a state s, if s occurs in μ. In the rest of the paper we assume, without loss of generality, that every *reachable* state admits an *executable* action a, i.e., $f(s, a) \neq \emptyset$. Consequently, every finite path can be extended into an infinite path. This allows us to consider the case of domains with infinite runs only. Obviously, when the assumption is not fulfilled, an additional action, say nop can be introduced.

A $\mathcal{D}$-path μ is said to be *fair* if for every state s and action a such that $s \xrightarrow{a}$ occurs infinitely many times in μ, it is the case that for every $s' \in f(s, a)$, $s \xrightarrow{a} s'$ occurs infinitely many times in μ. A $\mathcal{D}$*-trace* is a possibly infinite sequence $\tau = s_0 s_1 \ldots$ such that there exists some $\mathcal{D}$-path $\mu = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$. A $\mathcal{D}$-trace τ is said to be *fair* if there exists a $\mathcal{D}$-path $\mu' = s_0 \xrightarrow{a'_0} s_1 \xrightarrow{a'_1} \cdots$ that is so.

As it turns out, fairness captures the intuition that whenever an action is executed infinitely often, all of its effects take place infinitely often. This accounts for a form of nondeterminism that is typical when modeling *natural* events, as opposed to *adversarial* ones.

Planning languages such as STRIPS or ADL, all accommodated in the PDDL standard, are commonly used to specify the states and transitions in compact form. A *goal* in this language is a specification of the desired traces on $\mathcal{D}$. In particular, classical reachability goals, which require reaching a state s where a certain propositional formula φ over $Prop$ holds, are expressed as selecting all those *finite* traces $t = s_0 s_1 \cdots s_n$, such that $s_n \models \varphi$. Using *infinite traces* allows us to consider a richer set of goals, suitably expressed through LTL formulas.

2.2 Linear Temporal Logic (LTL)

LTL was originally proposed as a specification language for concurrent programs [Pnueli, 1977]. *Formulas* of LTL are built from a set $Prop$ of propositional symbols and are closed under the boolean operators, the unary temporal operators $\bigcirc$, $\Diamond$, and $\Box$, and the binary temporal operator $\mathcal{U}$.[1] Intuitively, $\bigcirc\varphi$ says that φ holds at the *next* instant, $\Diamond\varphi$ says that φ will *eventually* hold at some future instant, $\Box\varphi$ says that from the current instant on, φ will *always* hold, and $\varphi\,\mathcal{U}\,\psi$ says that at some future instant ψ will hold and *until* that point φ holds. We also use the standard boolean connectives $\vee$, $\wedge$, and $\rightarrow$.

The semantics of LTL is given in terms of interpretations over a *linear structure*. For simplicity, we use $\mathbb{N}$ as the linear structure: for an instant $i \in \mathbb{N}$, the successive instant is $i + 1$. An *interpretation* is a function $\iota : \mathbb{N} \rightarrow 2^{Prop}$ assigning to each element of $Prop$ a truth value at each instant $i \in \mathbb{N}$. For an interpretation ι, we inductively define when an LTL formula φ *is true* at an instant $i \in \mathbb{N}$ (written $\iota, i \models \varphi$):

- $\iota, i \models p$, for $p \in Prop$ iff $p \in \iota(i)$.
- $\iota, i \models \neg\varphi$ iff not $\iota, i \models \varphi$.
- $\iota, i \models \varphi \wedge \varphi'$ iff $\iota, i \models \varphi$ and $\iota, i \models \varphi'$.
- $\iota, i \models \bigcirc\varphi$ iff $\iota, i{+}1 \models \varphi$.
- $\iota, i \models \varphi\,\mathcal{U}\,\varphi'$ iff for some $j \geq i$, we have that $\iota, j \models \varphi'$ and for all k, $i \leq k < j$, we have that $\iota, k \models \varphi$.

A formula φ *is true* in ι (written $\iota \models \varphi$) if $\iota, 0 \models \varphi$. Given a planning domain, every trace $\tau = s_0 s_1 s_2 \ldots$ can be seen as an LTL interpretation ι such that $\iota, i \models p$ iff $s_i \models p$.

2.3 LTL and Büchi Automata

There is a tight relation between LTL and Büchi automata on infinite words [Vardi, 1996]. A *Büchi automaton* [Thomas, 1990] is a tuple $A = \langle \Sigma, Q, Q_0, \rho, F\rangle$, where:

- Σ is the finite input alphabet of the automaton;
- Q is the finite set of automaton states;
- $Q_0 \subseteq Q$ is the set of initial states of the automaton;
- $\rho : Q \times \Sigma \rightarrow 2^Q$ is the automaton transition function;
- $F \subseteq Q$ is the set of accepting states.

When Q_0 is a singleton, and $|\rho(q, \sigma)| \leq 1$, for every $q \in Q$ and $\sigma \in \Sigma$, A is said to be *deterministic*. When this is the case, we use q_0 instead of Q_0, and assume $\rho : Q \times \Sigma \rightarrow Q$.

[1] In fact, all operators can be defined in terms of $\bigcirc$ and $\mathcal{U}$.

The *input words* of A are the infinite words $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$. A *run* of A on an infinite word w is an infinite sequence of states $r = q_0q_1 \cdots \in Q^\omega$ such that $q_0 \in Q_0$ and $q_{i+1} \in \rho(q_i, \sigma_i)$. A run r is *accepting* iff $lim(r) \cap F \neq \emptyset$, where $lim(r)$ is the set of states that occur in r infinitely often. In other words, a run is accepting if it gets into F infinitely many times, which means, being F finite, that there is at least one state $q_f \in F$ visited infinitely often. The *language* accepted by A, denoted by $L(A)$, is the set of (infinite) words for which there is an accepting run.

The *nonemptiness* problem for an automaton A is to decide whether $L(A) \neq \emptyset$, i.e., whether the automaton accepts at least one word. The relevance of such problem for LTL follows from the correspondence obtained by setting the automaton alphabet to propositional interpretations, i.e., $\Sigma = 2^{Prop}$. Then, an infinite word over the alphabet 2^{Prop} represents an interpretation of an LTL formula over $Prop$. Formally, if the set of all models of an LTL formula φ is denoted as $M(\varphi)$, the result is that:

Theorem 1 [Vardi and Wolper, 1994] *For every* LTL *formula φ one can effectively construct a Büchi automaton A_φ whose number of states is at most exponential in the length of φ and such that $L(A_\varphi) = M(\varphi)$.*

In general, the automaton A_φ of Theorem 1 is *nondeterministic*. In fact, formulas exist for which no deterministic Büchi automaton accepts exactly its models, e.g., $\varphi = \Diamond\Box p$. Thus, nondeterministic Büchi automata are strictly more powerful than deterministic ones. In the following, we only consider LTL goal formulae φ such that A_φ is a deterministic Büchi automaton such that $L(A_\varphi) = M(\varphi)$. We call such formulas *deterministic*. Deterministic LTL formulas are general enough to capture many of the extended temporal goals and requirements that arise in practice.

3 The Problem

Given a (nondeterministic) planning domain $\mathcal{D} = \langle Act, Prop, S, S_0, f\rangle$, a *finite-state controller* (FSC) for $\mathcal{D}$ is a tuple $\Pi = \langle C, c_0, \Gamma, \Lambda, \delta, \Omega\rangle$, where:

- C is the finite set of controller states;
- $c_0 \in C$ is the initial controller state;
- $\Gamma = S$ is the controller input alphabet;
- $\Lambda = Act$ is the controller output alphabet;
- $\delta : C \times \Gamma \mapsto C$ is the controller transition function;
- $\Omega : C \mapsto \Lambda$ is the controller output function.

Formally, Π is a Moore machine whose input and output alphabets are the states and the actions of $\mathcal{D}$, respectively. The machine is intended to "drive" $\mathcal{D}$ so as to make it show a desired behavior. Note that, besides being able to observe the domain state, Π owns an internal state, allowing it to track past events, although in a limited way, as typical of finite-state machines.

A Π-*execution* is a possibly infinite sequence $\eta = c_0 \xrightarrow{s_0} c_1 \xrightarrow{s_1} \cdots$, such that $c_{i+1} = \delta(c_i, s_i)$. A $\mathcal{D}$-path $\mu = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ is said to be *induced* by Π if there exists a Π-execution $\eta = c_0 \xrightarrow{s_0} c_1 \xrightarrow{s_1} \cdots$ such that $a_i = \Omega(c_i)$. If so, Π is said to induce the $\mathcal{D}$-*trace* $\tau = s_0s_1 \ldots$. A state c of Π is Π-*reachable* if there exists a Π-execution where c occurs.

We are interested in those FSCs that, when executed on $\mathcal{D}$, yield only evolutions that satisfy a requirement expressed in the LTL goal. The following definitions capture this intuition. We say that a FSC Π *realizes an* LTL *goal* φ (on $\mathcal{D}$) if for all the $\mathcal{D}$-traces μ that Π induces on $\mathcal{D}$, it is the case that $\mu \models \varphi$. We say that Π *fairly* realizes an LTL goal φ (on $\mathcal{D}$) if for all the fair $\mathcal{D}$-traces μ induced by Π, $\mu \models \varphi$.

Thus, the notion of goal realization requires all evolutions of $\mathcal{D}$, obtained by executing Π, to satisfy φ. Observe that since the domain is nondeterministic, thus partially controllable, whenever Π prescribes the execution of an action in some state, it must also take care of all the possible outcomes, i.e., prescribing an action for each possible successor state.

An empty FSC inducing no $\mathcal{D}$-trace, realizes any LTL goal. To prevent such degenerate cases, we introduce the following notion. A FSC Π for a domain $\mathcal{D}$ is said to be *closed*, if: 1. $\Omega(c_0)$ and $\delta(c_0, s_0)$ are defined; 2. whenever $a = \Omega(c)$ and $c' = \delta(c, s)$ are defined, for some c and s, it is the case that $f(s, a) \neq \emptyset$, and, for every $s' \in f(s, a)$, $\Omega(c')$ and $\delta(c', s')$ are defined. In words, this definition requires Π to prescribe an *executable* action on both the initial state and on all the $\mathcal{D}$-states reachable by executing actions prescribed by Π. In the rest of the paper we consider only closed controllers. The problem of interest can thus be expressed as follows:

> Given a non-deterministic planning domain $\mathcal{D}$ and a deterministic LTL goal φ, build a closed finite-state controller Π that fairly realizes φ on $\mathcal{D}$.

We refer to this the problem as the LTL *fair realization problem* (for deterministic LTL goals over nondeterministic planning domains). This is a generalization of fully observable, non-deterministic planning problem (FOND), with reachability goals replaced by LTL goals, as well as an instance of the general formal synthesis problem for deterministic LTL formulas, where non-determinism is assumed to be fair. While we implicitly assume fairness, in synthesis it is typically stated in the input.

The standard approaches to controller synthesis are based on symbolic methods that compute the states over which certain types of "games" can be won. For example, the problem of realizing $\varphi = \Diamond p$ (eventually p), is mapped into that of finding the largest set S of states such that: S contains all the states satisfying p, and from any state in S, the system can be forced to reach a state satisfying p. If the initial state is in S, a strategy realizing φ can be obtained from the construction of S [Bloem *et al.*, 2012]. Similarly, to realize $\Box\Diamond p$ (always eventually p), one computes the set of states from which the system can be forced to make p true infinitely often. Such sets S, called *winning sets*, are characterized by fixpoint formulas in the μ-calculus (over game structures), and are typically computed by symbolic methods. By taking advantage of the symbolic representation, these methods can deal with very large collections of states (see, e.g., [Burch *et al.*, 1992]). Planning methods, on the other hand, are aimed at exploiting the structure of the planning domains by means of heuristics that operate on the compact representation of the domains.

We show below how to map the LTL fair realization problem into a planning problem. As a first step towards this goal, we show that we can restrict the search to controllers $\Pi = \langle C, c_0, S, Act, \delta, \Omega \rangle$ of a specific class, for which all components are fixed except for the *action selection function* Ω. Thus, the search for FSCs that achieve a goal φ with (deterministic) automaton A_φ will be reduced to the search for a *policy* Ω_φ mapping state-pairs $\langle q, s \rangle$ into actions where q is a Büchi automaton state and s is a domain state:

Theorem 2 *Consider a planning domain $\mathcal{D} = \langle Act, Prop, S, s_0, f \rangle$ and an* LTL *formula φ such that there exists a deterministic Büchi automaton $A_\varphi = \langle S, Q, q_0, \rho, F \rangle$ such that $L(A_\varphi) = M(\varphi)$. Then, there exists a FSC $\Pi = \langle C, c_0, S, Act, \delta, \Omega \rangle$ that fairly realizes φ on $\mathcal{D}$ iff there exists a FSC $\Pi_\varphi = \langle Q \times S, \langle q_0, s_0 \rangle, S, Act, \delta_\varphi, \Omega_\varphi \rangle$ that fairly realizes φ on $\mathcal{D}$, such that: $\delta_\varphi(q, s) = \langle q', s' \rangle$ iff $q' = \rho(q, s)$ and $s' \in f(s, \Omega_\varphi(q, s))$.*

Theorem 2 says that a FSC for realizing an LTL goal φ can be obtained from a policy $\pi = \Omega_\varphi$ over the domain $\mathcal{D}'$ that is the product of the non-deterministic domain $\mathcal{D}$ and the deterministic Büchi automaton A_φ. In the next section we show a theorem that characterizes the goal achieved by this policy over $\mathcal{D}'$, and the form in which such goal is achieved.

4 Reduction to Strong Cyclic Planning

A (nondeterministic) *planning problem* is a pair $P = \langle \mathcal{D}, G \rangle$, where $\mathcal{D}$ is a planning domain and $G \subseteq S$ a set of *goal* states. A *policy* π for a planning domain $\mathcal{D}$ is a function $\pi : S \to Act$, mapping states into actions. A $\mathcal{D}$-path μ is said to be *induced* by a policy π if $\mu = s_0 \xrightarrow{\pi(s_0)} s_1 \xrightarrow{\pi(s_1)} \cdots$. A $\mathcal{D}$-state s' is said to be π-*reachable* from a $\mathcal{D}$-state s if there exists a finite path $\mu = s_0 \xrightarrow{\pi(s_0)} \cdots \xrightarrow{\pi(s_{k-1})} s_k$ induced by π, such that $s_j = s$, for $0 \leq j \leq k$, and $s_k = s'$, for $k \geq 1$. A policy π is said to be a *non-terminating strong cyclic* solution to P if $\pi(s_0)$ is defined and for every state s π-reachable from s_0, there exists a state $s' \in G$ π-reachable from s.

The following result tells us that the output function Ω_A of the FSC Π_A of Theorem 2 can be easily extracted from a *non-terminating strong cyclic solution* π of a particular planning problem defined over the *cross-product domain* of $\mathcal{D}$ and A_φ.

Theorem 3 *Consider a planning domain $\mathcal{D} = \langle Act, Prop, S, s_0, f \rangle$ and an* LTL *formula φ such that there exists a deterministic Büchi automaton $A_\varphi = \langle S, Q, q_0, \rho, F \rangle$ with $L(A_\varphi) = M(\varphi)$. Let $\mathcal{D}' = \langle Act, Prop', S', s'_0, f' \rangle$ be the planning domain such that:*

- $Prop' = Prop \cup \{p_q \mid q \in Q\}$;
- $s'_0 = s_0 \cup \{p_{q_0}\}$;
- $f'(s, a) = s'_\mathcal{D} \cup p_{q'}$ *iff* $s'_\mathcal{D} \in f(s_\mathcal{D}, a)$ *and* $q' = \rho(q, s_\mathcal{D})$, *for* $s_\mathcal{D} = s \cap Prop$ *and* $p_q \in s$.

Further, let $G = \{s \in S' \mid p_q \in s, \ for \ q \in F\}$. Then, $\pi : S \to Act$ is a non-terminating strong cyclic solution *to $\langle \mathcal{D}', G \rangle$ iff the FSC $\Pi = \langle Q \times S, \langle q_0, s_0 \rangle, S, Act, \delta, \Omega \rangle$, with:*

- $\Omega(q, s_\mathcal{D}) = \pi(s)$ *for* $s = s_\mathcal{D} \cup p_q$;
- $\delta(q, s) = \langle q', s' \rangle$ *for* $q' = \rho(q, s)$, $s' \in f(s, \Omega(q, s))$;

fairly realizes φ on $\mathcal{D}$.

The theorem says that we can transform the problem of building a finite-state controller for an LTL goal into that of building a *non-terminating strong-cyclic policy* for a planning problem. We show next how to map the problem of computing a non-terminating strong cyclic policy for a non-deterministic problem $P = \langle \mathcal{D}, G \rangle$ into the problem of computing a *standard strong cyclic policy* for a non-deterministic problem $P' = \langle \mathcal{D}', G' \rangle$, which can be computed with any existing, off-the-shelf, strong cyclic planner. A (standard) *strong cyclic policy* is a policy π such that if s is a non-goal state π-reachable from s_0, then a goal state s' is π-reachable from s [Cimatti *et al.*, 2003]. The difference between standard (terminating) and non-terminating strong cyclic policies π is that the former can be *undefined* over the goal states, while the latter have to be defined over all π-reachable states, including goal states, as they must give rise to infinite executions.

Theorem 4 *A policy π is a* non-terminating strong cyclic solution *π to $P = \langle \mathcal{D}, G \rangle$ iff π is a* strong cyclic solution *to $P' = \langle \mathcal{D}', G' \rangle$, with $\mathcal{D}' = \langle Act, Prop', S', s_0, f' \rangle$ and $G' = \{s_g \cup \{ok\} \mid s_g \in G\}$, where:*

- $Prop' = Prop \uplus \{ok\}$;[2]
- $S' = S \cup G'$;
- $f'(s, a) = f(s, a) \cup \{s_g \cup \{ok\} \mid s_g \in f(s, a) \cap G\}$, *if* $f(s, a) \cap G \neq \emptyset$, *else* $f'(s, a) = f(s, a)$.

The transformation maps each goal state s_g in P into a non-goal state s_g of P'. In addition, a goal state $s_g \cup \{ok\}$ that is added as a possible successor state of the state-action pairs (s, a) that can transition into s_g. The transformation allows us to compute *non-terminating strong cyclic solutions* by resorting to planners for *(standard) strong cyclic solutions.* Notice that all the steps in such transformation involve only syntactic manipulations of the domain $\mathcal{D}$ and the Büchi automaton. Theorems 2–4 together result in the following theorem:

Theorem 5 *If there exists a* solution *to the fair realization problem defined by a non-deterministic planning domain $\mathcal{D}$ and a deterministic* LTL *formula φ, then a non-deterministic planning problem $P' = \langle \mathcal{D}', G' \rangle$ can be constructed such that the* strong cyclic solutions *to P' yield a FSC that fairly realizes φ on $\mathcal{D}$.*

5 Implementation

In line with Theorem 5, we have implemented a compiler that maps the compact description of a non-deterministic domain P and an LTL goal φ, into a standard strong cyclic planning problem P_φ in PDDL that can be solved with off-the-shelf planners. The first ingredient of the compilation is SPOT[3], an available tool able to map a large class of LTL goals φ into deterministic Büchi automata A_φ. We then take advantage of Theorems 3 and 4, for obtaining the strong cyclic planning problem P_φ by means of syntactic transformations of P and A_φ. P_φ is the cross-product of P and the propositional

[2] $\uplus$ stands for *disjoint union.*

[3] `http://spot.lip6.fr/ltl2tgba.html`

representation of the automaton A_φ, where an atom p_q is introduced to represent each state q of A_φ. The goal of P_φ is the dummy atom ok, inserted as an additional possible outcome of the actions that can add an atom p_q, for accepting state q of the automaton A_φ. The cross-product is encoded into two different ways, which give rise to two different but logically equivalent encodings of the strong cyclic planning problem P_φ. In the *sequential* encoding, domain actions are followed by actions that progress the state of the automaton A_φ, according to its transition function. In the *parallel* encoding, a new action is created for representing each possible sequence of a domain action followed by an automaton transition. In order to solve the resulting strong cyclic planning problem we use the recent PRP planner [Muise *et al.*, 2012], that like other recent solvers [Kuter *et al.*, 2008; Fu *et al.*, 2011], maps the strong cyclic planning problem into a sequence of classical planner calls.

6 Experiments

We have tested the proposed approach by running the PRP planner over the problems P_φ and comparing the method with the standard symbolic synthesis tool called TLV [Pnueli and Shahar, 1996], which provides an implementation of the algorithm for GR(1) synthesis described in [Bloem *et al.*, 2012].[4] TLV accepts the LTL goal φ directly along with a compact description of the non-deterministic domain obtained from P. In addition, we encode in LTL the assumptions about fairness so that the space of solutions of the two approaches coincide.

6.1 Lift

The *Lift controller* problem [Bloem *et al.*, 2012] requires to build a lift controller for an n-floor building, that guarantees every request from a floor to be eventually served. The main fluents we use to model the domain are at_i and req_i, one per floor ($i = 1, \ldots, n$), which represent, respectively, the lift being at floor i, and a request having been issued from floor i. As to actions, we use $push\_f_i\_1$, $push\_f_i\_2$, $move\_up\_from\_f_i$ ($i = 1, \ldots, n-1$), and $move\_down\_from\_f_i$ ($i = 2, \ldots, n$), with the following semantics: $push\_f_i\_1$ is executable only if $\neg at_i$, and nondeterministically either sets the value of req_i to $true$ or does nothing; $push\_f_i\_2$ is executable only when at_i holds, and sets the value of req_i to $false$; $move\_up\_from\_f_i$ unsets f_i and sets f_{i+1}; $move\_down\_from\_f_i$ unsets f_i and sets f_{i-1}. The lift is initially at floor 1, and no request is issued. We require move-up actions to be executable only if some request is issued. To this end we use a fluent $called$ in their precondition. To describe the domain evolution, we introduce auxiliary fluents that we use in combination with preconditions to force the alternate execution of a $move$ and n $push\_f_i$ actions, $i = 1, \ldots, n$. In this way, we capture all the possible situations the controller faces when the lift is at some floor. Notice that the nondeterminism of $push\_f_i\_1$ captures the fact that the lift may or may not be called from some floor. The LTL formula capturing the goal has the GR(1) form $\varphi = \bigwedge_{i=1}^{n} \Box\Diamond(req_i \rightarrow at_i)$. We generated 10 instances incrementing the number n of floors from 1 to 10.

[4] Also certain non-GR(1) problems, such as *Clerk* (see below), can easily be rewritten as GR(1).

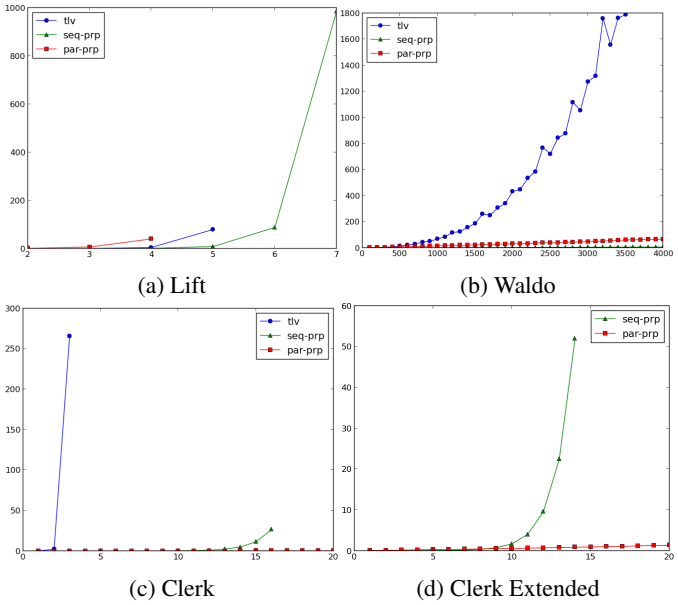
6.2 Waldo

The problem involves a robot that must move between locations until Waldo is found [Kress-Gazit *et al.*, 2007]. The robot moves in a circle of n rooms, and Waldo can appear non-deterministically (from nowhere) when the robot is in room i or $i/2$. When Waldo appears, the robot must remain in the same room. The robot must thus "patrol" rooms i and $i/2$ until Waldo appears. The LTL formula of this requirement is $\Box\Diamond(r_i \vee W) \wedge (r_{i/2} \vee W)$, where W represents the appearance of Waldo. In our account, the behavior of Waldo is *fair*, meaning that when the robot visits rooms i or $i/2$ infinitely often, Waldo will eventually show up (in fact, infinitely often). This is captured as non-deterministic effects in the actions that take the robot into rooms i and $i/2$. The result of the fairness assumption in this problem is that the best valid policy is to go either to i or $i/2$, move one step away and return to the same location. As in the original problem formulation the robot has to bounce between locations i and $i/2$ until Waldo appears, we considered a formulation with an extra predicate $visited_i$ to enforce that once the robot visits one target location, it has to visit the remaining ones before searching in the same location again. When every target location has been visited and Waldo didn't appear, the extra predicate $search\_again$ becomes true and all $visited_i$ false. The LTL goal is set to $\Box\Diamond(search\_again \vee W)$. We generated 40 instances, incrementing the number n of rooms by 100, from 100 up to 4000.

6.3 Clerk

This domain models the actions of a clerk in a store. The problem requires to build a controller which guarantees that every client request is served. New customers show up requesting one of the n items or packages p_i, which can be available or not. If the item is not available, the clerk buys the product from a supplier first. The available items p_i are described by the fluents $instore\_p_i$ and the package the customer requests by fluent $want\_p_i$. If no customer is requesting an item, the action $request$ will nondeterministically make true one of the $want\_p_i$ fluents and the fluent $item\_requested$. If $instore\_p_i$ is true, the action $sell\_p_i$ renders $instore\_p_i$ and $want\_p_i$ to be false while making true $item\_served$. Otherwise, $buy\_supply\_p_i$ adds $instore\_p_i$ when $want\_p_i$ is true and $instore\_p_i$ false. The goal is $\Box(item\_requested \rightarrow \Diamond item\_served)$.

We tested also an extended version of the same domain where package p_i must be stored and retrieved in a cell loc_i of a $1 \times (n+1)$ grid. The desk of the clerk is at loc_{n+1}. If a package is requested, the clerk has to find the package in the grid, $pick$ it up and $sell$ it to the customer at the desk. If a supplier brings a package, the clerk has to pick the package and $store$ it in its correct location. The goal then is encoded with the formula $\Box(active\_request \rightarrow \Diamond(item\_served \vee item\_stored))$, where $active\_request$ is made true every time the action $request$ is applied, and made false once the request has been satisfied, either stored



(a) Lift (b) Waldo

(c) Clerk (d) Clerk Extended

Figure 1: Comparison of TLV and PRP over sequential and parallel encoding where y-axis stands for time in seconds and x-axis for number of (a), rooms (b), and packages (c,d)

or served. Instances were generated for the two versions of the domain with number n of packages ranging from 1 to 20.

6.4 Results

All experiments were run on an Intel Core i7-3770 3.4GHz processor, with a timeout of 30 minutes and a memory limit of 4GB. The results are shown in Figure 1. In **Lift**, TLV is unable to solve 6 floors, while PRP solves up to 7 floors. The sequential encoding scales up better than the parallel one, which runs out of memory while parsing the problem with 5 floors. In **Waldo**: TLV solves 35 instances, taking 1788.6s to solve the instance with 3500 rooms, while PRP solves all 40 instances, taking 6.8s and 66.72s in the largest instance with 4000 rooms in the sequential and parallel encodings respectively. Finally, in **Clerk**, PRP solves the complete suite of benchamarks up to 20 packages with the parallel encoding, each instance in less than 0.25s. With the sequential encoding, PRP solves up to 16 floors in 26.54s, running out of memory in the larger instances. TLV fails even with 4 packages, and for 3 it takes 265s (against 0.04 of PRP). In the extended version of the domain, PRP based on parallel encoding solves all instances in the suite in less than 1.4s, while all instances up to 14 packages in 52.01s, when using the sequential encoding. TLV solves a few instances only, and is not shown.

The results of the experiments show that the planning approach scales up better than TLV, in particular, as the problems become more planning-like with many actions and propositions. The encoding of the cross-product between the planning domain and Büchi automaton, that has been implemented in two ways, sequential and parallel, makes a large difference in some cases, with the parallel encoding performing better as the size of the Büchi automaton decreases. Last, the LTL goal can often be encoded in many different ways too. In many of these domains there is an enumerative approach with goals like 'if $request_i$ comes alive, then perform $serve_i$' and a generic approach like 'if there is request, then serve it', which can be rendered equivalent by tinkering with the planning domain. When this is possible, smaller automata are obtained, which benefits the planning approach further.

Finally, we performed tests on *deterministic* problems to compare with the approach reported recently in [Patrizi *et al.*, 2011], where *deterministic* planning with general LTL goals is mapped into a classical planning problem that is solved once, and whose solution encodes a lasso-plan that achieves the LTL goal. For this, we considered their *Gripper* instances. In terms of time, both approaches performed similarly, with PRP being slightly faster on average (0.38s vs. 0.4s), and finding more compact policies. In terms of coverage, out of 500 instances, PRP solved 470, 70 more problems than the other approach, that time out in compilation. It is worth noting that our strong-cyclic-planning approach does not reduce to the classical approach in [Patrizi *et al.*, 2011] over deterministic problems. This is because the proposed mapping makes the resulting problem P_φ non-deterministic even if the original domain P is deterministic (this is the result of the transformation in Theorem 4 for mapping non-terminating strong cyclic policies into standard ones). Yet the approach appears to be competitive with the deterministic approach when this is used for computing lasso-plans.

7 Conclusions

We have considered the problem of planning in environments where the state is fully observable, actions have (fair) non-deterministic effects, and plans must generate infinite state trajectories for complying with LTL goals. The problem is a generalization of the fully-observable non-deterministic planning problem (FOND) where reachability goals are replaced by temporally extended LTL goals, and a special case of the more general formal synthesis problem where non-determinism is assumed fair and LTL formulas encode deterministic Büchi automata. We have shown that the problem can be compiled into a strong cyclic planning problem that can be solved by off-the-shelf planners. The experiments show that the approach is computationally meaningful and has potential benefits in relation to standard symbolic synthesis methods. The proposed formulation extends the scope of current planning methods, which can thus be used effectively to generate controllers for systems that must operate continuously, under exogenous interventions that cannot be predicted or controlled.

Acknowledgments

This work was partly supported by the EU projects FP7-ICT 257593 (ACSI) and EC-7PM-SpaceBook, and the ARC project LP110100115.

References

[Albarghouthi *et al.*, 2009] A. Albarghouthi, J. Baier, and S. McIlraith. On the use of planning technology for verification. In *Proc. ICAPS'09 Workshop VV&PS*, 2009.

[Alur and La Torre, 2004] Rajeev Alur and Salvatore La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.

[Bacchus and Kabanza, 1998] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Math. and AI*, 22:5–27, 1998.

[Baier and McIlraith, 2006] J.A. Baier and S.A. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *Proc. AAAI'06*, 2006.

[Baier *et al.*, 2009] J.A. Baier, F. Bacchus, and S.A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Art. Int.*, 173(5-6), 2009.

[Bloem *et al.*, 2012] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[Burch *et al.*, 1992] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[Bylander, 1994] Tom Bylander. The Computational Complexity of Propositional STRIPS Planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, 2003.

[Cresswell and Coddington, 2004] S. Cresswell and A. Coddington. Compilation of LTL goal formulas into PDDL. In *Proc. ECAI'04*, 2004.

[Daniele *et al.*, 1999] Marco Daniele, Paolo Traverso, and Moshe Y. Vardi. Strong Cyclic Planning Revisited. In *ECP*, pages 35–48, 1999.

[De Giacomo and Vardi, 1999] G. De Giacomo and M. Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Proc. ECP'99*, 1999.

[Edelkamp, 2003] S. Edelkamp. Promela planning. In *Proc. SPIN'03*, 2003.

[Edelkamp, 2006] S. Edelkamp. On the compilation of plan constraints and preferences. In *ICAPS'06*, 2006.

[Fu *et al.*, 2011] Jicheng Fu, Vincent Ng, Farokh B. Bastani, and I-Ling Yen. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proc. of IJCAI*, pages 1949–1954, 2011.

[Gerevini and Long, 2005] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. Technical report, Univ. of Brescia, 2005.

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

[Kress-Gazit *et al.*, 2007] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Proc. of ICRA*, pages 3116–3121, 2007.

[Kuter *et al.*, 2008] Ugur Kuter, Dana S. Nau, Elnatan Reisner, and Robert P. Goldman. Using Classical Planners to Solve Nondeterministic Planning Problems. In *Proc. of ICAPS*, pages 190–197, 2008.

[Morgenstern and Schneider, 2008] Andreas Morgenstern and Klaus Schneider. From LTL to Symbolically Represented Deterministic Automata. In *Proc. of VMCAI*, pages 279–293, 2008.

[Morgenstern and Schneider, 2011] Andreas Morgenstern and Klaus Schneider. A LTL Fragment for GR(1)-Synthesis. In *Proc. of iWIGP*, pages 33–45, 2011.

[Muise *et al.*, 2012] Christian J. Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proc. of ICAPS*, 2012.

[Patrizi *et al.*, 2011] Fabio Patrizi, Nir Lipovetzky, Giuseppe De Giacomo, and Hector Geffner. Computing infinite plans for LTL goals using a classical planner. In *Proc. of IJCAI*, pages 2003–2008, 2011.

[Pnueli and Rosner, 1989] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proc. of POPL*, pages 179–190, 1989.

[Pnueli and Shahar, 1996] Amir Pnueli and Elad Shahar. A platform for combining deductive with algorithmic verification. In *Proc. of CAV*, pages 184–195, 1996.

[Pnueli, 1977] Amir Pnueli. The Temporal Logic of Programs. In *Proc. of FOCS*, pages 46–57, 1977.

[Richter and Westphal, 2010] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39:127–177, 2010.

[Thomas, 1990] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science.* Elsevier, 1990.

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

[Vardi, 1996] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency*, volume 1043 of *LNCS*. Springer, 1996.