Bridging the Gap Between LTL Synthesis and Automated Planning[*]

Abstract

Linear Temporal Logic (LTL) synthesis can be understood as the problem of building a controller that defines a winning strategy, for a two-player game against the environment, where the objective is to satisfy a given LTL formula. It is an important problem with applications in software synthesis, including controller synthesis. Recent work has explored the close connection between automated planning and LTL synthesis but has not provided a full mapping between the two problems nor have its practical implications been explored. In this paper we establish the correspondence between LTL synthesis and fully observable non-deterministic (FOND) planning. We also provide the first explicit compilation that translates an LTL synthesis problem to a FOND problem. Experiments with state-of-the-art LTL FOND and synthesis solvers show automated planning to be a viable and effective tool for highly structured LTL synthesis problems.

1 Introduction

The problem of synthesizing software, including controllers, from logical specification is a fundamental problem in AI and computer science more generally. Church's synthesis problem was first posed by Church in 1957 in the context of synthesizing digital circuits from a logical specification [Church, 1957] and is considered one of the most challenging problems in reactive systems [Piterman *et al.*, 2006]. Two common approaches to solving the problem have emerged: reducing the problem to the emptiness problem of tree automata, and characterizing the problem as a two-player game.

In 1989, Pnueli and Rosner examined the problem of reactive synthesis using Linear Temporal Logic (LTL) [Pnueli, 1977] as the specification language (what we refer to here as "LTL synthesis") viewing the problem as a two-player game, and showing that this problem was 2EXPTIME-complete [Pnueli and Rosner, 1989]. Over the years, this discouraging result has been mitigated by the identification of several restricted classes of LTL for which the complexity of the synthesis problem need not be so high (e.g., [Asarin *et al.*, 1998; Alur and La Torre, 2004]). More recently Piterman, Pnueli, and Sa'ar examined the synthesis of reactive designs when the LTL specification was restricted to the class of so-called *Generalized Reactivity(1)* (GR1) formulae, presenting an N^3-time algorithm which checks whether the formula is realizable, and in the case where it is, constructs an automaton representing one of the possible implementing circuits [Piterman *et al.*, 2006]. Today, a number of synthesis tools exist with varying effectiveness (e.g., Acacia+ [Bohy *et al.*, 2012], Lilly [Jobstmann and Bloem, 2006]).

Recent work has explored connections between automated planning and synthesis (e.g., [De Giacomo *et al.*, 2010; Sardiña and D'Ippolito, 2015; De Giacomo and Vardi, 2015]) but has not provided a full mapping between the two problems, nor have the practical implications of such a mapping been explored from an automated planning perspective. In this paper we investigate the relationship between (LTL) synthesis and automated planning, and in particular (LTL) Fully Observable Non-Deterministic (FOND) planning. We do so by leveraging a correspondence between FOND and 2-player games. This work is inspired by significant recent advances in the computational efficiency of FOND planning that have produced FOND planners that scale well in many domains (e.g., NDP [Alford *et al.*, 2014], FIP [Fu *et al.*, 2011], MYND [Mattmüller *et al.*, 2010] and PRP [Muise *et al.*, 2012]). Our insights are that just as SAT can be (and has been) used as a black-box solver for a myriad of problems that can be reduced to SAT, so too can FOND be used as a black-box solver for suitable problems. Establishing the connection between FOND and 2-player games not only provides a connection to LTL synthesis – the primary subject of this exploration – it also provides the key to leveraging FOND for other problems.

In Section 3 we establish the correspondence between LTL synthesis and strong solutions to FOND planning. This is followed in Section 4 by the first approach to automatically translate a realizability problem, given by an LTL specification, into a planning problem, described in the Planning Domain Definition Language (PDDL), the de facto standard input language for automated planners. Experiments with state-of-the-art LTL synthesis and FOND solvers illustrate that the choice of formalism and solver technology for a problem can have a dramatic impact. We elucidate some of the properties that would indicate why one technique should be used over the other. As a general rule-of-thumb, if the problem is highly structured and the uncertainty largely restricted, planning-based approaches will excel. Such highly structured problems are evident in synthesis problems for physical devices.

2 Preliminaries

2.1 FOND

A FOND planning problem is tuple $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$, where $\mathcal{F}$ is a set of *fluents*; $\mathcal{I} \subseteq \mathcal{F}$ characterizes what holds ini-

[*]Under review.

tially; $\mathcal{G} \subseteq \mathcal{F}$ characterizes what must hold for the goal to be achieved; and $\mathcal{A}$ is the set of actions. The set of literals of $\mathcal{F}$ is $Lits(\mathcal{F}) = \mathcal{F} \cup \{\neg f \mid f \in \mathcal{F}\}$.

Each action $a \in \mathcal{A}$ is associated with $\langle Pre_a, Eff_a \rangle$, where $Pre_a \subseteq Lits(\mathcal{F})$ is the precondition and Eff_a is a set of outcomes of a. Each outcome $e \in Eff_a$ 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 one outcome e in Eff_a, s' is equal to $s \setminus \{f \mid (C \rightarrow \neg f) \in e, s \models C\} \cup \{f \mid (C \rightarrow f) \in e, s \models C\}$. A *policy* p, is a partial function from states to actions such that if $p(s) = a$, then a is applicable in s. An *execution* π of a policy p in state s is a sequence $s_0, a_0, s_1, a_1, \ldots$ (either finite or infinite), where $s_0 = s$, and such that every state-action-state substring 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.

A finite execution π *achieves* a set of literals L if its ending state s is such that $s \models L$. An infinite execution π *achieves* a set of literals L if there exists a state s that appears infinitely often in π and that is such that $s \models \mathcal{G}$. 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 [Geffner and Bonet, 2013]. Note this implies that finite executions are fair. A policy p is a *strong-cyclic plan* for a FOND problem $P = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$, iff every fair execution of p over $\mathcal{I}$ satisfies the goal. A policy p is a *strong plan* for P iff every execution of p over $\mathcal{I}$ satisfies $\mathcal{G}$.

2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) is a propositional logic extended with temporal modal operators. The set of LTL formulae over a set of propositions $\mathcal{P}$ is defined inductively as follows. p is a formula if $p \in \mathcal{P}$ or the constant $\top$. If φ_1 and φ_2 are LTL formulas, then so are $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, and $\circ\varphi_1$ and $\varphi_1 \mathsf{U} \varphi_2$. Let σ be an infinite sequence of subsets of $\mathcal{P}$, and φ be an LTL formula. Then σ *satisfies* φ, denoted as $\sigma \models \varphi$ iff $\sigma, 0 \models \varphi$, where:

- $\sigma, i \models p$, for each $p \in \mathcal{P} \cup \{\top\}$ iff $s_i \models p$.
- $\sigma, i \models \neg\varphi$ iff $\sigma, i \models \varphi$ does not hold.
- $\sigma, i \models \varphi_1 \wedge \varphi_2$ iff $\sigma, i \models \varphi_1$ and $\sigma, i \models \varphi_2$.
- $\sigma, i \models \circ\varphi$ iff $\sigma, (i+1) \models \varphi$.
- $\sigma, i \models \varphi_1 \mathsf{U} \varphi_2$ iff there exists a $j \geq i$ such that $\sigma, j \models \varphi_2$, and $\sigma, k \models \varphi_1$, for each $k \in \{i, i+1, \ldots, j-1\}$.

Additional constants and operators are defined as follows $\bot \equiv \neg\top$, $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\Diamond\varphi \equiv \top \mathsf{U} \varphi$, $\Box\varphi \equiv \neg\Diamond\varphi$.

Given an LTL formula φ, it is possible to construct Nondeterministic Büchi Automaton (NBA) A_φ that accepts σ iff $\sigma \models \varphi$ [Vardi and Wolper, 1994]. The size of the NBA is worst-case exponential in the size of φ.

2.3 LTL FOND

Recently Camacho *et al.* [2017] extended FOND with LTL goals. An LTL FOND problem is a tuple $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$, where all elements are like before, but where $\mathcal{G}$ is a formula over $\mathcal{F}$. In short, LTL executions are defined just like in FOND, and a policy is a strong-cyclic (resp. strong) plan for problem P if each fair (resp. unrestricted) execution π is such that, when removing all actions from π it results in a sequence of states σ such that $\sigma \models \mathcal{G}$.

2.4 LTL Synthesis

The LTL synthesis problem [Pnueli and Rosner, 1989] intuitively describes a two-player game between a controller and the environment. The game consists of an infinite sequence of turns. In each turn the environment chooses an action, and the controller then chooses another. Each action actually corresponds to setting the values of some variables. The controller has a winning strategy if, no matter how the environment plays, the sequences of states generated satisfies a given LTL formula φ.

Formally, a synthesis problem is a tuple $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, where disjoint sets of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, the environment variables, and $\mathcal{Y} = \{y_1, \ldots, y_m\}$, the controller variables. An LTL formula over $\mathcal{X} \cup \mathcal{Y}$ is *realizable* if there exists a function $f : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that for every infinite sequence of subsets of $\mathcal{X}$, $X_1X_2\ldots$, it holds that $\pi = (X_1 \cup f(X_1)), (X_2 \cup f(X_1X_2)), (X_3 \cup f(X_1X_2X_3))\ldots$ is such that $\pi \models \varphi$. Intuitively, no matter what the choice of the environment is, which is given by the sequence $X_1X_2\ldots$, the controller has a strategy, given by f, that ensures formula φ is satisfied in the resulting game. The *synthesis* problem corresponds to actually computing function f.

3 Relationship Between FOND and Synthesis

Both LTL synthesis and FOND are related to two-player games: in both problems an agent (or controller) seeks a solution that achieves a condition no matter what choices are taken by the environment. There are however two important differences. First, in LTL synthesis the controller reacts to the environment; in other words, the environment "plays first", while the controller "plays second". Rather, in FOND, the play sequence is inverted since the environment decides the outcome of an action, which is in turn defined by the agent (controller). Second, state-of-the-art FOND solvers find strong-cyclic solutions, and indeed those types of solutions are considered standard. This assumes fairness in the environment, which is not an assumption inherent to LTL synthesis. Thus a correct mapping between FOND and Synthesis should handle fairness correctly.

Previous work has explored the relation between FOND and Synthesis. Sardiña and D'Ippolito [2015] show how to translate FOND as a reactive synthesis problem by expressing fairness constraints as temporal logic formulae. De Giacomo and Vardi [2013] sketches a mapping from FOND to LTL synthesis, in which the effects of actions are specified using LTL. This approach, however, does not dive into the details of the inverted turns. Neither do the works by De Giacomo *et al.*; Kissmann and Edelkamp [2010; 2009], which show a correspondence between two-player game structures and FOND planning.

In the rest of the section we provide an explicit mapping between LTL FOND and LTL Synthesis. We aim at a correct mapping between both problems. Efficiency is the focus of the next section.

To establish a correspondence between LTL synthesis and LTL FOND, we address the inverted turns by considering the negation of realizability. Observe that an instance $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is not realizable iff for every function $f : (2^{\mathcal{X}}) \to 2^{\mathcal{Y}}$ there exists a sequence $X_1 X_2 X_3 \ldots$ of subsets of $\mathcal{X}$, such that:

$$X_1 \cup f(X_1), X_2 \cup f(X_1 X_2), X_3 \cup f(X_1 X_2 X_3) \ldots \models \neg\varphi$$

Note that what comes after the "iff" maps directly into an instance of LTL FOND. Indeed, we define the problem $P_\varphi = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ such that fluents are the union of all variables: $\mathcal{F} = \mathcal{X} \cup \mathcal{Y}$, the set of actions is the set of subsets of $\mathcal{X}$: $\mathcal{A} = \{A_x \mid x \subseteq \mathcal{X}\}$. Intuitively action A_x is always executable (has empty preconditions) and deterministically sets to true the variables in x and to false the variables in $\mathcal{X} \setminus x$. In addition, it non-deterministically sets the values of variables in $\mathcal{Y}$ to every possible combination. Formally, $\mathit{Eff}_{A_x} = \{e_{x,y} \mid y \subseteq \mathcal{Y}\}$, where each $e_{x,y} = \{f \mid f \in x \cup y\} \cup \{\neg f \mid f \in (\mathcal{X} \cup \mathcal{Y}) \setminus (x \cup y)\}$. Finally, we set $\mathcal{I} = \{\}$ and $\mathcal{G} = \bigcirc\neg\varphi$.

Theorem 1. *$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is realizable iff P_φ has no strong plan.*

In the other direction, let $P = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ be an LTL FOND problem. We now construct a synthesis problem $\langle \mathcal{X}_P, \mathcal{Y}_P, \varphi_P \rangle$ following well-known encodings of planning into SAT [Rintanen *et al.*, 2006]; the difference now is that we use LTL to establish a connection between a state and its successors, instead of different variables, and that we consider explicitly that actions have a number of outcomes.

For each action in $a \in \mathcal{A}$, we create a variable $a \in \mathcal{X}_P$. Each fluent $f \in \mathcal{F}$ is also a variable in $\mathcal{X}_P$. Variables in $\mathcal{Y}_P$ are used to choose one of the non-deterministic outcomes of each action; this way if the action with the largest number of outcomes has n outcomes, we create $\lceil \log n \rceil$ variables, whose objective is to "choose" the outcome for an action. To model the preconditions of the action, we conjoin in φ_P, for each action a the formula $\Box(a \to \bigwedge_{\ell \in Pre_a} \ell)$. We express the fact that only one action can execute at a time by conjoining to φ_P the formulae $\Box \bigvee_{a \in \mathcal{A}} a$, and $\Box(a \to \neg a')$, for each $a' \in \mathcal{A}$ different from a. To model the fact that the environment selects the outome being performed, for each action outcome e we create a variable a_e in $\mathcal{X}_P$. For each action $a \in \mathcal{A}$ and outcome $e \in \mathit{Eff}_a$, we add formulae of the form $\Box a \wedge \chi_{a,e} \to a_e$, where $\chi_{a,e}$ is a formula over $\mathcal{Y}_P$, which intuitively "selects" outcome e for action a. For space, we do not go into the details of how to encode $\chi_{a,e}$. However, these formulae have the following property: for any action a, given an assignment for $\mathcal{Y}_P$ variables there is exactly one $e \in \mathit{Eff}_a$ for which $\chi_{a,e}$ becomes true. This models the fact that the $\mathcal{Y}_P$ variables are used to select the outcomes.

Finally, we now conjoin to φ_P formulae to express the dynamics of the domain. Specifically we add successor-state-axiom-like expressions [Reiter, 2001] of the form:

$$\Box(\bigcirc f \equiv (\phi_f^+ \vee (f \wedge \neg\phi_f^-))), \quad \text{for each } f \in \mathcal{F}$$

where ϕ_f^+ is a formula that encodes the conditions under which f becomes true after an outcome has occurred, and where ϕ_f^- encodes the conditions under which f becomes false in the next state. Both of these formulae can be computed from Eff_a [Reiter, 2001]. Finally, we conjoin to φ_P the fluents in the initial state $\mathcal{I}$, and the goal formula, $\mathcal{G}$

Now, it is not hard to see that there exists a strong solution to the LTL problem P iff there exists a sequence of settings of the $\mathcal{X}_P$ variables, $X_1, X_2, \ldots$ such that for every sequence of settings of the $\mathcal{Y}$ variables, $Y_1, Y_2, \ldots$

$$(X_1 \cup Y_1), (X_2 \cup Y_2), (X_3 \cup Y_3), \ldots \models \varphi_P$$

The correspondence is established by the following result.

Theorem 2. *Let $P = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ be an LTL FOND problem. P has a strong plan iff $\langle \mathcal{X}_P, \mathcal{Y}_P, \neg\varphi_P \rangle$ is not realizable.*

4 Approach

In Section 3 we established the correspondence between existence of solutions to LTL synthesis, and existence of strong solutions to LTL FOND planning. In this section we introduce the first translation from LTL synthesis into FOND planning (and by inclusion, into LTL FOND).

4.1 Compiling LTL Synthesis to FOND

Our approach to solve an LTL synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ as FOND consists of three stages. First, we pre-process $\mathcal{P}$. Second, we compile it into a standard FOND problem $\mathcal{P}'$. Finally, we solve $\mathcal{P}'$ with a strong-cyclic planner. Extracting a strategy for $\mathcal{P}$ from a solution to $\mathcal{P}'$ is straightforward, and we omit the details for lack of space.

In a pre-processing stage, we first simplify the specification, if possible, by removing from $\mathcal{X}$ and $\mathcal{Y}$ those variables that do not appear in φ (cf. Section 4.2). Then, we transform φ into an NBA $A_\varphi = (Q, \Sigma, \delta, q_0, Q_{Fin})$, where Q is the set of automaton states, Σ is the alphabet, δ is the transition function, q_0 is the initial state of the automaton, and $Q_{Fin} \subseteq Q$ is the set of accepting states. We assume transitions $T \in \delta$ have guard formula $guard(T) = \bigvee_m c_m$ specified in DNF. We use `spot` [Duret-Lutz *et al.*, 2016] to perform such transformation. We consider the NBA that results from replacing each transition $T = \delta(q, q')$ with a set of transitions $\{T_m\}_m$ from q to q', each with guard c_m that is a conjunction of literals in $\mathcal{X} \cup \mathcal{Y}$. Certainly, the new NBA is also an NBA for φ as it contains the same runs. In the rest of this section, we use A_φ to refer to the new NBA with these transitions T_m.

In a second stage, we compile $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ into a FOND problem $\mathcal{P}'(h_{max}) = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ that depends on a parameter, $h_{\max}$, that intuitively sets a search horizon. The sets of fluents, $\mathcal{F}$, and actions, $\mathcal{A}$, of the problem are listed below. In what follows, we describe the dynamics of the problem.

$$\begin{aligned}
\mathcal{F} = &\{q, q^S, q^T, q^{S,T} \mid q \in Q\} \cup \{goal\} \cup \\
&\{env\_mode, aut\_mode, can\_switch, can\_accept\} \cup \\
&\{at\_horizon(h)\}_{0 \le h \le h_{max}} \cup \{next(h{+}1, h)\}_{0 \le h < h_{max}} \\
&\{turn_k\}_{1 \le x \le |\mathcal{X}|} \cup \{v_x, v_{\neg x}\}_{x \in \mathcal{X}} \cup \{v_y, v_{\neg y}\}_{y \in \mathcal{Y}} \\
\mathcal{A} = &\{move\_k\}_{1 \le k \le |\mathcal{X}|} \cup \{trans(T) \mid T \in \delta\} \cup \\
&\{switch2aut, switch2env, accept\}
\end{aligned}$$

Environment Mode In the environment mode, the dynamics of the problem simulates the move of the environment, which is uncontrollable by the agent. The environment can play $2^{|\mathcal{X}|}$ different moves. In order to reduce the branching factor of the problem, we simulate the move with a cascade of non-deterministic actions, each one setting (v_x) or unsetting ($v_{\neg x}$) the value of a variable x in $\mathcal{X}$.

$$\begin{aligned}
Pre_{move\_k} &= \{env\_mode, turn_k\} \\
\mathit{Eff}_{move\_k} &= oneof(\{v_{xk}\}, \{v_{\neg xk}\}) \cup \Psi_{move\_k} \\
\Psi_{move\_k} &= \begin{cases} \{turn_{k+1}, \neg turn_k\}, & \text{if } k < |\mathcal{X}| \\ \{can\_switch, \neg turn_k\}, & \text{if } k = |\mathcal{X}| \end{cases}
\end{aligned}$$

After the environment move has been simulated, the *switch2aut* action switches to the automaton mode, where automaton states will be progressed according to the current automaton configuration. To do so, the automaton configuration (represented by fluents of the form q and q^T) are *frozen* into copies q^S and $q^{S,T}$. It is important to notice that the compilation captures, in a single planning state, multiple runs of the automaton. Intuitively, if fluent q in a planning state is true then there exists a run of the automaton finishing in q.

$$\begin{aligned}
Pre_{switch2aut}(h,h') &= \{env\_mode, can\_switch\} \cup \\
&\qquad \{at\_horizon(h), next(h',h)\} \\
\mathit{Eff}_{switch2aut}(h,h') &= \{at\_horizon(h'), \neg at\_horizon(h)\} \\
&\cup \{aut\_mode, \neg env\_mode, \neg can\_switch\} \cup \\
&\{q \rightarrow \{q^S, \neg q\}, q^T \rightarrow \{q^{S,T}, \neg q^T\} \mid q \in Q\}
\end{aligned}$$

Automaton Mode The automaton mode simulates, in parallel, the assignment to variables in $\mathcal{Y}$ and the transitions in the automaton configuration. Whereas the update in the automaton configuration is usually understood as a *response* to the observation to variables in $\mathcal{X} \cup \mathcal{Y}$, the dynamics of the problem takes a different perspective: the agent can decide which automaton transitions wants to perform, and then sets the value to variables in $\mathcal{Y}$ so that the transition guards are satisfied. Such transitions are simulated by means of $trans(T)$ actions, one for each $T \in \delta$. The advantage of taking this perspective is that the preconditions of the $trans(T)$ actions capture the *relevant* subset of uncontrollable variables needed to perform automaton transitions, and only the value of a subset of variables in $\mathcal{Y}$, the *relevant* ones, is set. As we discuss in Section 4.2, exploiting relevance in planning has potential advantages in terms of performance.

$$\begin{aligned}
Pre_{trans(T)} &= \{aut\_mode, q_i^S\} \cup \{\neg v_{\neg l} \mid l \in guard(T)\} \\
\mathit{Eff}_{trans(T)} &= \{q_j\} \cup \{v_l \mid l \in guard(T)\} \cup \Psi_{trans(T)} \\
\Psi_{trans(T)} &= \begin{cases} \left\{q_i^{S,T} \rightarrow q_j^T\right\}, & \text{if } q_j \notin Q_{Fin} \\ \{can\_accept\}, & \text{if } q_j \in Q_{Fin} \end{cases}
\end{aligned}$$

Fluents v_l simulate the truth value of variables in $\mathcal{X} \cup \mathcal{Y}$. More precisely, v_x (resp. $v_{\neg x}$) simulates that $x \in \mathcal{X}$ is true (resp. false), and similarly for v_y and $v_{\neg y}$ We abuse notation and write $l \in guard(T)$ if the literal l appears (not negated) in $guard(T)$. As usual, we use the equivalence $\neg(\neg l) = l$.

Note that there are automaton fluents *tokenized* with a superindex T (of the form q^T and $q^{S,T}$). Intuitively, the token T in an automaton state q indicates that the agent commits to transition the runs that finish in q into runs that visit an accepting state. This notion is captured by $\Psi_{trans(T)}$. The conditional effects $q_i^{S,T} \rightarrow q_j^T$ do *not* delete $q_i^{S,T}$, and this is required to allow the agent to perform more than one automaton transition (Recall that NBA's are non-deterministic).

$$\begin{aligned}
Pre_{switch2env}(h,h') &= \{aut\_mode\} \\
\mathit{Eff}_{switch2env}(h,h') &= \{env\_mode, \neg aut\_mode, \} \cup \\
&\qquad \{turn_1\} \cup Regularize \\
Regularize &= \{q^S \rightarrow \neg q^S, q^{S,T} \rightarrow \neg q^{S,T} \mid q \in Q\} \cup \\
&\quad \{\neg v_x, \neg v_{\neg x} \mid x \in \mathcal{X}\} \cup \{\neg v_y, \neg v_{\neg y} \mid y \in \mathcal{Y}\}
\end{aligned}$$

The agent can, at any time in the automaton mode, execute *switch2env* to switch to the environment mode. Special predicates $at\_horizon(h)$ are used to monitor how many steps away from a recognized accepting state the current planning state is. The purpose of *Regularize*, which is optional, is to improve the search performance as described in Section 4.2.

Finally, the *accept* action lets the planner recognizably reach accepting states in the automaton, with the purpose of finding accepting runs (i.e., runs that visit accepting states infinitely often). By executing the *accept* action, the agent sets tokens q^T to commit to finding a run of the automaton that progresses q into an accepting state – at which point the token q^T will be deleted, as per the dynamics of the transition actions $trans(T)$. When such a commitment has not been made, the action effects prune those runs of the automaton that did not visit an accepting state, meaning that the corresponding runs do not need to be extended into accepting runs.

$$\begin{aligned}
Pre_{accept}(h) &= \{aut\_mode, can\_accept, at\_horizon(h)\} \\
\mathit{Eff}_{accept}(h) &= oneof(\{goal\}, \\
&\{Regularize \cup \{at\_horizon(0), \neg at\_horizon(h)\} \cup \\
&\{env\_mode, \neg aut\_mode, \neg can\_accept\} \cup \\
&\{q \rightarrow q^T \mid q \in Q\} \cup \{q^T \rightarrow \{\neg q^T, \neg q^S\} \mid q \in Q\}\}
\end{aligned}$$

Initial and Goal States The initial state of the problem is $\mathcal{I} = \{q_0, env\_mode, turn_1\} \cup \{at\_horizon(0)\} \cup \{next(h+1, h) \mid h \in 0..h_{max}\}$. The goal is $\mathcal{G} = \{goal\}$.

Definition 1 (Syn2FOND compilation). *For an* LTL *synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, and parameter h_{max}, the* Syn2FOND compilation *transforms φ into an NBA and constructs the FOND problem $\mathcal{P}'(h_{max}) = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ as described above.*

The fluents $at\_horizon(h)$ in $\mathcal{P}'$ make it possible to simulate an adversarial game with the environment, where the agent is forced to reach, recognizably, an accepting state within h_{max} moves. The non-determinism introduced by the action *accept* is such strong-cyclic plans that are solution to $\mathcal{P}'$ yield infinite plan executions, because the dummy goal is non-existent in $\mathcal{P}$ and therefore never reached. Rather, infinite plans are produced that reach accepting states infinitely often, and therefore satisfy the LTL specification. For more details, we refer the reader to [Camacho *et al.*, 2017].

Finally, our algorithm to solve LTL synthesis, as an iterated search of solutions to $\mathcal{P}'(h)$, $h = 1, 2 \ldots$ is sound and complete. Intuitively, the result comes from the dynamics of

Syn2FOND described above, and the finite size of the search space of automaton configurations.

Lemma 1 (soundness)**.** *Strong-cyclic plans to a Syn2FOND compiled FOND problem $\mathcal{P}'$ yield solutions to $\mathcal{P}$.*

Lemma 2 (completeness)**.** *An* LTL *synthesis problem $\mathcal{P}$ is realizable iff the compiled planning problem $\mathcal{P}'(h_{max})$ has a strong-cyclic solution for some finite h_{max}.*

Theorem 3 (correctness)**.** *The iterated search for strong-cyclic solutions to the FOND problem $\mathcal{P}'(h)$, with $h = 1, 2, \ldots$ eventually finds a solution to $\mathcal{P}'(h)$, if one solution to the* LTL *synthesis $\mathcal{P}$ exists.*

4.2 Exploiting Relevance

The exploitation of relevance has been extensively studied within automated planning as a means of improving the efficiency of plan generation and execution monitoring (e.g., [Haslum *et al.*, 2013]). Here, we present a suite of ideas in a similar vein that leverages state relevance in service of efficient synthesis via FOND planning.

Global Irrelevance We say that a set of variables $Z \subseteq \mathcal{X} \cup \mathcal{Y}$ is *globally irrelevant* wrt φ if, for every infinite sequence $\pi = \{Z_n\}_n$ of assignments to variables in $\mathcal{X} \cup \mathcal{Y}$, for every k, and for every subset $Z' \subseteq Z$, it holds that $\pi \models \varphi$ iff $\pi' \models \varphi$, where $\pi' = \{Z'_n\}_n$ is such that $Z'_n = Z_n$ if $n \neq k$, and $Z'_k = (Z_k \setminus Z) \cup Z'$. In words, the truth of variables in $\mathcal{Z}$ in an infinite sequence π does not influence whether π satisfies the specification φ.

Theorem 4. *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ be an* LTL *synthesis problem, and let $vars(\varphi) \subseteq \mathcal{X} \cup \mathcal{Y}$ be the set of variables that appear in φ. Then, $Z = (\mathcal{X} \cup \mathcal{Y}) \setminus vars(\varphi)$ is a globally irrelevant set of variables.*

Corollary 1. *The pre-process stage in Syn2FOND prunes a set of globally irrelevant variables in $\mathcal{X} \cup \mathcal{Y}$ wrt φ.*

An immediate consequence of Theorem 4 – which follows trivially from the observation that variables in Z do not appear in the specification φ – is that the pruning of (the globally irrelevant set of) variables $Z = (\mathcal{X} \cup \mathcal{Y}) \setminus vars(\varphi)$ performed by the Syn2FOND compilation does not affect the correctness of the approach. As we show in the experimental section, reducing the number of variables in the problem – even in the naive way that Syn2FOND does – makes it, in general, more tractable, and not doing so may induce problems of scalability when the sets of globally irrelevant variables increase.

Commonly used techniques in automated planning reason about goal reachability – exploiting the compact description of world change based on the preconditions and effects of the actions –, and prune variables (and also actions) that are not relevant to the achievement of the goal (cf. [Helmert, 2006; Palacios and Geffner, 2007]). By solving a synthesis problem as planning, we can benefit from variable pruning techniques existing in automated planning tools.

Local Irrelevance The dynamics of the Syn2FOND compilation, presented in Section 4, makes it possible to prune what we can intuitively consider to be locally irrelevant variables at planning time. In particular, when the agent decides to apply a $trans(T)$ action, only variables in $guard(T)$ are implicitly considered to be relevant. This has two benefits. First, by computing plans that only consider a subset of the uncontrollable variables in $\mathcal{X}$, the planner reduces the exponential blowup (in $|\mathcal{X}|$) in the search caused by all possible environment's moves. Second, by assigning values to a reduced subset of the controllable variables in $\mathcal{Y}$ (only those that are necessary to transition in the NBA accordingly), the planner is able to compute more compact policies that are conditioned on only what is relevant.

State Regularization The states regularization performed by the *switch2env* and *accept* actions leverage the fact that, at the next planning step, the value of the variables $v_x, v_{\neg x}, v_y, v_{\neg y}$ (that simulate the variables in $\mathcal{X}$ and $\mathcal{Y}$, is not relevant. By abstracting all planning states, the search space is notably reduced. Besides savings in search space, this also reduces the planning run time, because it is more likely for a strong-cyclic planner to find a loop in the abstract space than in the concrete space.

5 Evaluation

Our main objective for the evaluation is to give a sense of when to choose one formalism over another. Although the same problems can be represented as either LTL synthesis or FOND planning, the choice of formalism can have a dramatic impact on the time required to find a solution. We would expect the FOND setting to be better suited for problems with more "structure", and our results serve to illustrate this.

We consider four natural sources of problem encodings: (1) problems encoded directly as LTL synthesis; (2) problems encoded directly as FOND planning; (3) problems from (2) that have been converted to the LTL synthesis setting following the proposal in [De Giacomo and Vardi, 2013] modified to better capture the FOND setting; and (4) problems from (1) that have been converted automatically using the method described in Section 4. A fifth alternative is to encode a problem as a mixture of synthesis and FOND, following a setting such as LTL-FOND extended to the synthesis task with uncontrollable variables. Our encoding in Section 4 can be naturally extended to this setting, and we defer empirically investigating this option for future work.

In our experiments, we used state-of-the-art synthesis and FOND tools Acacia+ [Bohy *et al.*, 2012] and PRP [Muise *et al.*, 2012]. We explicitly note the source of encodings (corresponding to the four options above), and the LTL synthesis problems are specified using the TLSF format [Jacobs *et al.*, 2016]. For the conversion from LTL synthesis to FOND, we have created an automated tool that will be released along with the publication of this work. For the opposite direction, we manually converted the FOND problems to the TLSF format using the technique described above.

First, we consider some representative problems from both synthesis and FOND perspectives. Table 1 presents the time to compute solutions for both Acacia+ and PRP. The first group of problems (lilly and load) come from the synthesis community and are encoded in TLSF and automatically converted by our tool to FOND. The second group of problems (tire, chain, and path) come from the FOND benchmark tire-

	LTL Synthesis (Acacia+)			FOND (PRP)	
problem	Aut	Syn	Total	Search	Total
lilly-p5	0.03	0.01	**0.04**	0.12	2.58
lilly-p6	0.00	0.04	**0.04**	0.28	3.94
lilly-p7	0.04	0.00	**0.04**	0.08	0.56
load-p2	0.08	0.02	**0.10**	23.74	264.98
load-p3	0.13	0.11	**0.24**	59.54	450.86
load-p4	0.22	2.17	**2.39**	time	—
tri-tire-p3	time	—	—	0.01	**0.01**
chain-p2	time	—	—	0.01	**0.01**
path-p2	time	—	—	0.01	**0.01**
build-3	0.52	0.15	0.67	0.0	**0.06**
build-4	3.02	2.38	5.40	0.02	**0.08**
build-5	49.92	67.73	117.65	0.12	**1.06**
irr-build-2	8.39	33.56	41.95	0.02	**0.32**
irr-build-3	15.54	97.22	112.76	0.02	**1.24**
irr-build-4	24.82	728.39	753.21	0.04	**3.52**

Table 1: Performance (measured in seconds) of LTL synthesis and FOND planning tools. (Aut) Time to compile the specification into automata. (Syn) Time for synthesis. (Search) Time taken during search for a solution in PRP (remaining time is bookkeeping and policy maintenance).

world, and were converted to LTL synthesis. The final set of problems is a new domain that we introduce, and encode directly (and as compactly as possible) in both LTL and FOND.

The first thing to note is the drastic performance hit that can occur converting from one formalism to another. Going from LTL synthesis to FOND is workable in some instances, but the opposite direction proved impossible for even the simplest of problems. This is because the proposed translation requires complex constraints to properly maintain the reachable state-space. It is this "structure" that we conjecture the synthesis tools struggle with, and test separately below.

The **build** domain addresses the problem of building maintenance, and requires the agent to maintain which rooms have their lights on or off depending on the time of day and whether or not people are in the room. The environment controls the transition between day and night, as well as when people enter or leave a room. The agent must control the lights in response. We strove to build the most compact and efficient model in TLSF and FOND for a fair comparison.

Each subsequent **build-#** adds another room, and we can see that the synthesis tools scale far worse in this aspect (both in generating automata and performing the synthesis). Each **irr-build-#** problem introduces # rooms that may non-deterministically be vacuumed at night (controlled by the environment). Note that this is irrelevant to the task of turning lights on or off (the vacuuming can be done in any lighting condition). The relevance reasoning present in the FOND planner is able to cope with this by largely ignoring the irrelevant aspects of the environment, but the synthesis component of Acacia+ struggles a great deal. This highlights the strength of the FOND tools for leveraging state relevance to solve problems efficiently.

Finally, we created a synthetic domain that lets us tune the level of "structure" in a problem: more structure leads to fewer possibilities for the environment to act without violating a constraint or assumption. In the **switches** domain, a total of n switches, $s_1 \dots s_n$, initially switched on, need to be all switched off eventually. The environment affects the state of the switches non-deterministically. However, the dynamics of the environment is such that immediately after the agent switches off s_k, the environmental non-determinism can only affect the state of a certain number of switches $s_{k'}$, with $k' > k$. A trivial strategy for the agent is to switch off s_1 to s_n in that order.

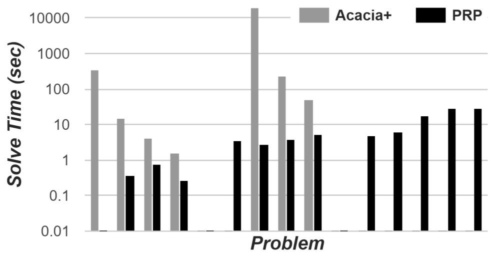

Figure 1: Run time (in seconds) for the switch domain.

We encoded a series of *switches* problems, natively as LTL specifications in TLSF format and also as FOND. Figure 1 shows how Acacia+ and PRP coped with the range of problems. They are in three distinct sets (each of increasing number of switches), and within each group the problems range from most structured to least (by varying k).

The problems in the first two groups are solved quite readily by PRP, and so the trend is less clear, but for the larger problems we find that PRP struggles when there is less structure and the environment can flip many switches without violating an assumption. On the other side, we find that the most structured problems are also the most difficult for Acacia+, and the synthesis becomes easier when there is less structure (i.e., more switches can be flipped).

The structure we tune in the switches domain is one property of a problem that may favour FOND technology over the synthesis tools. Another is the presence of irrelevance we discuss in Section 4.2 and surfaced in the variation of the building maintenance benchmark. Other notions, such as causal structure in the problem, may also play an important role in features that separate the effectiveness of the two formalisms. We plan to investigate these possibilities in future work.

6 Concluding Remarks

LTL synthesis is both an important and challenging problem for which broadly effective tools have remained largely elusive. Motivated by recent advances in the efficiency of FOND planning, this work sought to examine the viability of FOND planning as a computational tool for the realization of LTL synthesis. To this end, we established the theoretical correspondence between LTL synthesis and strong solutions to FOND planning. We also provided the first approach to automatically translate a realizability problem, given by an LTL specification, into a planning problem described in PDDL. Experiments with state-of-the-art LTL synthesis and FOND solvers highlighted properties that challenged or supported each of the solvers. Our experiments show automated planning to be a viable and effective tool for highly strucutred LTL synthesis problems.

References

[Alford *et al.*, 2014] Ron Alford, Ugur Kuter, Dana Nau, and Robert P Goldman. Plan aggregation for strong cyclic planning in nondeterministic domains. *Artificial Intelligence*, 216:206–232, 2014.

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

[Asarin *et al.*, 1998] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proceedings of the IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, 1998.

[Bohy *et al.*, 2012] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 652–657, 2012.

[Camacho *et al.*, 2017] Alberto Camacho, Eleni Triantafillou, Christian Muise, Jorge A. Baier, and Sheila A. McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[Church, 1957] Alonzo Church. Applications of recursive arithmetic to the problem of circuit synthesis. *Summaries of the Summer Institute of Symbolic Logic, Cornell University 1957*, 1:3–50, 1957.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1558–1564, 2015.

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Paolo Felli, Fabio Patrizi, and Sebastian Sardiña. Two-player game structures for generalized planning and agent composition. In Maria Fox and David Poole, editors, *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2010.

[Duret-Lutz *et al.*, 2016] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on on Automated Technology for Verification and Analysis (ATVA)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016.

[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 *Proceedings of the 22nd International Joint Conference On Artificial Intelligence (IJCAI)*, pages 1949–1954, 2011.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. A Concise Introduction to Models and Methods for Automated Planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(2):1–141, 2013.

[Haslum *et al.*, 2013] Patrik Haslum, Malte Helmert, and A Jonsson. Safe, strong and tractable relevance analysis for planning. In *23rd International Conference on Automated Planning and Scheduling*, pages 317–321, 2013.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Jacobs *et al.*, 2016] Swen Jacobs, Felix Klein, and Sebastian Schirmer. A high-level LTL synthesis format: TLSF v1.1. In *Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016.*, pages 112–132, 2016.

[Jobstmann and Bloem, 2006] Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *Formal Methods in Computer-Aided Design, 6th International Conference, FMCAD 2006, San Jose, California, USA, November 12-16, 2006, Proceedings*, pages 117–124, 2006.

[Kissmann and Edelkamp, 2009] Peter Kissmann and Stefan Edelkamp. Solving fully-observable non-deterministic planning problems via translation into a general game. In *Proceedings of the 32nd Annual German Conference on AI (KI09)*, pages 1–8, 2009.

[Mattmüller *et al.*, 2010] Robert Mattmüller, Manuela Ortlieb, Malte Helmert, and Pascal Bercher. Pattern database heuristics for fully observable nondeterministic planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 105–112, 2010.

[Muise *et al.*, 2012] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved Non-deterministic Planning by Exploiting State Relevance. In *Proceedings of the 22nd International Conference on Automated Planning and Sched. (ICAPS)*, pages 172–180, 2012.

[Palacios and Geffner, 2007] Hector Palacios and Hector Geffner. From Conformant into Classical Planning: Efficient Translations that May Be Complete Too. In *17th International Conference on Automated Planning and Scheduling*, pages 264–271, 2007.

[Piterman *et al.*, 2006] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings*, pages 364–380, 2006.

[Pnueli and Rosner, 1989] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190, 1989.

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

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, MA, 2001.

[Rintanen *et al.*, 2006] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

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

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