Finite LTL Synthesis as Planning **

Alberto Camacho[†], Jorge A. Baier[‡], Christian Muise[⋆], Sheila A. McIlraith[†]
[†]Department of Computer Science, University of Toronto.
[‡]Pontificia Universidad Católica de Chile, and Chilean Center for Semantic Web Research.
[⋆]IBM Research. Cambridge Research Center. USA.
[†]{acamacho,sheila}@cs.toronto.edu, [‡]jabaier@ing.puc.cl, [⋆]christian.muise@ibm.com

Abstract

LTL synthesis is the task of generating a strategy that satisfies a Linear Temporal Logic (LTL) specification interpreted over infinite traces. In this paper we examine the problem of $\mathsf{LTL_f}$ synthesis, a variant of LTL synthesis where the specification of the behaviour of the strategy we generate is interpreted over *finite* traces – similar to the assumption we make in many planning problems, and important for the synthesis of business processes and other system interactions of finite duration. Existing approaches to $\mathsf{LTL_f}$ synthesis transform $\mathsf{LTL_f}$ into *deterministic* finite-state automata (DFA) and reduce the synthesis problem to a DFA game. Unfortunately, the DFA transformation is worst-case double-exponential in the size of the formula, presenting a computational bottleneck. In contrast, our approach exploits *non-deterministic* automata, and we reduce the synthesis problem to a non-deterministic planning problem. We leverage our approach not only for strategy generation but also to generate certificates of unrealizability – the first such method for $\mathsf{LTL_f}$. We employ a battery of techniques that exploit the structure of the $\mathsf{LTL_f}$ specification to improve the efficiency of our transformation to automata. We combine these techniques with lazy determinization of automata and on-the-fly state abstraction. We illustrate the effectiveness of our approach on a set of established LTL synthesis benchmarks adapted to finite LTL.

1 Introduction

Synthesizing software from logical specification is a fundamental problem in computer science dating back to Alonzo Church (Church 1957). In 1989, Pnueli and Rosner examined the problem of reactive synthesis, proposing the use of Linear Temporal Logic (LTL) as a specification language, and showing that so-called *LTL synthesis* is 2EXPTIME-complete (Pnueli and Rosner 1989).

Traditional approaches to LTL synthesis rely on transforming the LTL specification into *deterministic* automata, for which a so-called *winning region* is computed. Computing the winning region is polynomial in the size of the deterministic automaton. However, computing such an automaton is worst-case double-exponential in the size of the LTL formula, and this becomes a computational bottleneck in the synthesis process. To mitigate for this, Acacia$^+$, the state of the art in LTL synthesis, transforms the LTL formula into multiple worst-case exponential *non-deterministic Büchi automata* (NBA). Acacia$^+$'s computation of the winning region implicitly performs a determinization of the automaton, and thus is worst-case double-exponential. However, it has the computational advantage that the determinization is symbolic, and in practice it only instantiates a reduced subset of the states (Bohy et al. 2012).

Our concern in this paper is with the synthesis of LTL specifications interpreted over *finite* traces ($\mathsf{LTL_f}$ synthesis) (De Giacomo and Vardi 2015). Many interesting problems of finite duration, including typical planning problems, and business processes, are most appropriately specified with respect to a finite interpretation. Indeed, the study of LTL interpreted over finite traces (henceforth $\mathsf{LTL_f}$) dates back within the planning community, at least, to TLPlan (Bacchus and Kabanza 2000), and a subset of $\mathsf{LTL_f}$ can be found in the specification of PDDL 3.0 (Gerevini et al. 2009). The correspondence between $\mathsf{LTL_f}$ and automata has similarly been exploited within automated planning for well over a decade in support of tasks such as planning with temporally extended goals (e.g. (Baier and McIlraith 2006; Edelkamp 2006)) and preferences (e.g., (Baier, Bacchus, and McIlraith 2009; Coles and Coles 2011)). Perhaps most similar to $\mathsf{LTL_f}$ (resp. LTL) synthesis is the task of planning with $\mathsf{LTL_f}$ (resp. LTL) goals in non-deterministic environments (e.g. (Patrizi, Lipovetzky, and Geffner 2013; Camacho et al. 2017)). Both tasks aim to synthesize controllers that guarantee satisfaction of the $\mathsf{LTL_f}$ (resp. LTL) formula. In contrast, the dynamics of a planning task is characterized by a highly structured explicit action model that captures non-determinism from the environment as uncertain action effects, whereas in synthesis the problem is reduced to repeated assignments of variables by either the agent or environment. Preliminary studies of the connection between LTL synthesis and planning have been recently presented (Camacho et al. 2018).

Like its infinite counterpart, $\mathsf{LTL_f}$ synthesis is 2EXPTIME-complete (De Giacomo and Vardi 2015) and existing approaches to $\mathsf{LTL_f}$ synthesis rely on transformations of the $\mathsf{LTL_f}$ specification into a *deterministic finite-state automaton* (DFA) (De Giacomo and Vardi 2015; Zhu et al. 2017). DFA transformations are worst-case double-exponential in the size of the $\mathsf{LTL_f}$ formula, matching the complexity of $\mathsf{LTL_f}$ synthesis, and become the computational bottle neck of these approaches. Very recently, the first $\mathsf{LTL_f}$ synthesis implementation was

** Correction to published version: see footnote on page 7.

developed, *Syft*, which computes the winning region over a symbolic representation of the DFA rather than as an explicit graph (Zhu et al. 2017).

In this paper, we present the first $\mathsf{LTL_f}$ synthesis system that not only computes strategies but that also produces certificates of unrealizability – a strategy and proof certificate that will cause the the $\mathsf{LTL_f}$ specification to be violated. Such certificates are very useful for incremental design and debugging, for verification, and as a tool for optimization. In contrast to existing theory and to the only other $\mathsf{LTL_f}$ realization we are aware of, our system relies on a transformation to NFA (rather than DFA). We then reduce $\mathsf{LTL_f}$ synthesis to the task of finding a strong plan for a *fully-observable non-deterministic* (FOND) planning problem. The FOND problem captures the dynamics of *non-deterministic finite-state automata* (NFA) that corresponds to the original $\mathsf{LTL_f}$ formula. Transformation of an LTL formula into an NFA is worst-case exponential in the size of the formula, and hence is computationally more appealing than a worst-case double-exponential DFA transformation. The compilations in our implemented system employ a battery of techniques that allow planning algorithms to exploit structure, including symbolic automata decompositions of the specification, *lazy* determinization of the NFA, and on-the-fly state *abstractions*.

The rest of the paper is organized as follows. We first review $\mathsf{LTL_f}$ and its correspondence with automata, and introduce the FOND planning model (Section 2). We then give a formal definition of $\mathsf{LTL_f}$ synthesis (Section 3) and define what stands for a certificate of unrealizability (Section 4). We exploit the duality between these two problems in a technique to determine realizability and unrealizability (Section 5). We establish a bidirectional mapping between $\mathsf{LTL_f}$ synthesis and FOND planning (Section 6). The following sections describe the algoithmic details of our reductions from $\mathsf{LTL_f}$ realizability and unrealizability to FOND planning. Finally, we discuss the results of an empirical evaluation of our algorithms implemented in a tool that we called *SynKit*, and we close with concluding remarks.

2 Preliminaries

2.1 Linear Temporal Logic

Linear Temporal Logic (LTL), first introduced by Pnueli (1977) as a specification language for reactive synthesis, is a propositional modal logic with modalities referring to time. $\mathsf{LTL_f}$ has essentially the same syntax as LTL but is interpreted over finite traces. In particular, given a set of propositional symbols, P, $\mathsf{LTL_f}$ formula φ is defined as follows:

$$\varphi := \top \mid \bot \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

where $p \in P$, and $\bigcirc$ (*next*) and U (*until*) are temporal operators. Intuitively, the *next* specifies what needs to hold in the next time step, and the *until* specifies what needs to hold at least until something else holds. Other temporal operators such as *eventually* ($\Diamond$), *always* ($\Box$), and *release* (R) are defined by the standard equivalences: $\Diamond\varphi \equiv \top \,\mathsf{U}\, \varphi$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$, and $\varphi_1 \,\mathsf{R}\, \varphi_2 \equiv \neg(\neg\varphi_1 \,\mathsf{U}\, \neg\varphi_2)$. $\mathsf{LTL_f}$ also has a *weak-next* operator ($\bullet$), defined by $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$, that tells that φ needs to hold in the next time step if such next time step exists. Note that $\neg\bigcirc\varphi \not\equiv \bigcirc\neg\varphi$ (De Giacomo and Vardi 2013). Within the planning community, the study of LTL interpreted over *finite* traces dates back at least to the work by Bacchus and Kabanza (1998) on specifying temporally extended goals, and was incorporated into PDDL 3.0 in 2006 (Gerevini et al. 2009). Recent work uses $\mathsf{LTL_f}$ (De Giacomo and Vardi 2013) as a specification language (e.g. (Camacho et al. 2017; De Giacomo et al. 2017)).

$\mathsf{LTL_f}$ formulae are interpreted over finite traces $\sigma = s_0 \cdots s_n$ of propositional states, where each s_i is a set of propositions from $\mathcal{P}$ that are true in s_i. We say that σ *satisfies* an $\mathsf{LTL_f}$ formula φ, denoted $\sigma \models \varphi$, when $\sigma, 0 \models \varphi$, 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 $\sigma, i \models \varphi_2$.
- $\sigma, i \models \bigcirc\varphi$ iff $i < n$ and $\sigma, (i+1) \models \varphi$.
- $\sigma, i \models \varphi_1 \,\mathsf{U}\, \varphi_2$ iff there exists a $i \leq j \leq n$ such that $\sigma, j \models \varphi_2$, and $\sigma, k \models \varphi_1$, for each $i \leq k < j$.

The interpretation of other modal operators such as *eventually* and *always* follow from their definitions.

2.2 $\mathsf{LTL_f}$ and Finite State Automata

For every $\mathsf{LTL_f}$ formula φ, it is possible to construct a *nondeterministic finite-state automaton* (NFA), A_φ, in worst-case exponential time, which accepts the models of φ (Baier and McIlraith 2006). Such an NFA is a tuple $A_\varphi = \langle Q, \Sigma, q_0, \delta, \alpha \rangle$, where Q is a finite set of states, Σ contains all subsets of propositions in φ, $q_0 \in Q$ is the initial state, $\delta \subseteq Q \times L(P) \times Q$ is a transition relation, where $L(P)$ is the set of propositional formulae over P, and $\alpha \subseteq Q$ is a set of accepting states.

A *run* of A_φ on a *word* $w = x_1 \cdots x_n \in \Sigma^*$ is a sequence of states $q_0 q_1 \cdots q_n$ such that $(q_{i-1}, \varphi_i, q_i) \in \delta$ and $x_i \models \varphi_i$ for each $i \in \{1, \ldots, n\}$. A run is *accepting* if $q_n \in \alpha$. In general, more than one run of A_φ may exist for an given input word. A word w is accepted when *some* run of A on w is accepting. The *language* of A is the set of words accepted by A. Sometimes we slightly abuse notation and denote by $\delta(q, x)$ the set of states q' for which there is a formula ψ such that $(q, \psi, q') \in \delta$ and $x \models \varphi$.

An NFA is, in particular, a *deterministic finite-state automaton* (DFA) when exactly one run exists for every input string. A DFA can be constructed (e.g. with the well-known *powerset construction*) by determinizing an NFA (Rabin and Scott 1959).

2.3 FOND Planning

A FOND planning problem is a tuple $\mathcal{P} = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$, where $\mathcal{F}$ is a finite set of propositional fluents; $\mathcal{I} \subseteq \mathcal{F}$ describes what holds in the *initial state*; $\mathcal{G} \subseteq \mathcal{F}$ describes what must hold in a state for the goal to be achieved; and $\mathcal{A}$ is a finite set of actions. We denote by $Lits(\mathcal{F}) := \mathcal{F} \cup \{\neg f \mid f \in \mathcal{F}\}$ the set of *literals* of $\mathcal{F}$. Planning *states* are sets of literals, and we say that a state s *satisfies* a literal f, denoted by $s \models f$, iff $f \in s$.

Each action $a \in \mathcal{A}$ is described by a set of *preconditions* (Pre_a) and a set of *effects* (Eff_a). The preconditions $Pre_a \subseteq Lits(\mathcal{F})$ describe what must hold in a state s for a to be applicable. The outcome of a is selected nondeterministically from one of the effects in Eff_a. Elements e in Eff_a are sets of tuples $c_i \rightarrow e_i$. Each e_i is a set of literals that describes the outcome s' of a relative to state s and condition c_i. Each c_i is a set of literals that describes the *condition* that must hold in a state s for e_i to have effect. When no conditions exist, we simply write effects $e \in \mathit{Eff}_a$ as sets of literals. Formally, $s' \models f$ iff $s \models f$ and $\neg f \notin e$, or $f \in e$. Sometimes we abuse notation and write $\mathit{oneof}(p, q) \in \mathit{Eff}_a$ to denote that each effect in Eff_a must be interpreted as two effects that contain, respectively, literals p and q.

A *policy* is a mapping π from states to actions such that, if $\pi(s) = a$, then a is applicable in s. A finite execution of π from state s_0 is a finite sequence of state-actions $s_0, a_0, s_1, a_1, \ldots s_n$ where each s_{i+1} is an outcome of s_i by a_i. A finite execution of π *achieves* the goal $\mathcal{G}$ if $s_n \models f$ for each $f \in \mathcal{G}$.

Different classes of solutions to FOND problems have been studied. In particular, *strong solutions* are policies whose execution guarantees goal achievement in a finite number of steps, despite non-determinism. The class of *strong cyclic solutions* are policies π where the goal can be achieved by π from each state that is reachable by π from the initial state. It has been shown that strong planning and strong cyclic planning are EXPTIME-complete problems (Rintanen 2004).

3 LTL-f Realizability and Synthesis

The problem of LTL synthesis was first introduced by Pnueli and Rosner (1989) to study the synthesis of a reactive module where the specification language used was LTL. Recently, De Giacomo and Vardi (2015) studied the synthesis problem for $\mathsf{LTL_f}$ specifications (cf. Definition 1), and determined the problem to be 2EXPTIME-complete – and therefore matching the complexity of LTL synthesis. In what follows, we denote by $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ the $\mathsf{LTL_f}$ synthesis problem with specification φ over *uncontrollable* (resp. *controllable*) variables $\mathcal{X}$ (resp. $\mathcal{Y}$).

Definition 1 ($\mathsf{LTL_f}$ realizability and synthesis)**.** *Given two disjoint sets of variables, $\mathcal{X}$ and $\mathcal{Y}$, the* realizability *problem for an $\mathsf{LTL_f}$ specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is to determine whether there exists a strategy $f : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that, for each infinite sequence $\{X_i\}_{i \geq 0} \in (2^{\mathcal{X}})^{\omega}$ of subsets of $\mathcal{X}$, the sequence $\{X_i \cup f(X_0 \cdots X_i)\}_{i \geq 0}$ has a* finite *prefix that satisfies φ. In this case, f is said to be a* winning *strategy. The* synthesis *problem for a realizable $\mathsf{LTL_f}$ specification is to compute a winning strategy. When $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is not realizable, we say that it is* unrealizable*.*

As noted briefly in the Section 1, existing approaches to $\mathsf{LTL_f}$ synthesis rely on transformation of the $\mathsf{LTL_f}$ specification into a DFA. These approaches then reduce the problem to finding a solution to a so-called DFA game where the *agent* player controls the $\mathcal{X}$ variables, the *environment* player controls the $\mathcal{Y}$ variables, and the objective is for the agent player to generate a state trajectory that is accepting with respect to the DFA (De Giacomo and Vardi 2015; Zhu et al. 2017). Solving a DFA game can be done in polynomial time in the size of the DFA. The computational bottle neck of these approaches has been the transformation of the specification into a DFA, which is worst-case double-exponential in the size of the $\mathsf{LTL_f}$ formula and matches the complexity of $\mathsf{LTL_f}$ synthesis.

$\mathsf{LTL_f}$ synthesis can be reduced to LTL synthesis by translating the original $\mathsf{LTL_f}$ specification formula φ into a larger LTL formula φ' that contains extra variables and operators, and then solving the resulting LTL synthesis problem. Unfortunately, the resulting LTL formula φ' is prohibitively large, causing the resulting transformation to a corresponding Büchi automata to be impossible in practice. By comparison, transformation of the original $\mathsf{LTL_f}$ formula, φ to a corresponding DFA is much easier to compute. Zhu et al. show that their state-of-the-art tool for $\mathsf{LTL_f}$ synthesis, *Syft*, outperforms the approach of synthesis via reduction to LTL synthesis.

4 LTL-f Unrealizability

In this section we define $\mathsf{LTL_f}$ certificates of unrealizability, and establish the correspondence between $\mathsf{LTL_f}$ realizability and the existence of unrealizability certificates.

An $\mathsf{LTL_f}$ synthesis problem is unrealizable when no winning strategy for the agent exists. Viewed another way, the problem is unrealizable when the *environment*, rather than the agent, has a winning strategy in a similar setting. Formally, $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is unrealizable when the environment has a strategy $g : (2^{\mathcal{Y}})^* \rightarrow 2^{\mathcal{X}}$ such that, for any sequence $\{Y_i\}_{i \geq 0} \in (2^{\mathcal{Y}})^{\omega}$, the *infinite* sequence $(g(\epsilon) \cup Y_0), \{g(Y_0 \cdots Y_i) \cup Y_{i+1}\}_{i \geq 0}$ does *not* have a finite prefix that satisfies φ. That is, g is such that *every* finite prefix of $(g(\epsilon) \cup Y_0), \{g(Y_0 \cdots Y_i) \cup Y_{i+1}\}_{i \geq 0}$ satisfies $\neg\varphi$. Here, we write $g(\epsilon) \subseteq \mathcal{X}$ to denote the first move of the environment, which does not depend on any of the agent moves.

Definition 2. *A* certificate *of unrealizability for an $\mathsf{LTL_f}$ specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is a strategy $g : (2^{\mathcal{Y}})^* \rightarrow 2^{\mathcal{X}}$ such that, for any sequence $\{Y_i\}_{i \geq 0} \in (2^{\mathcal{Y}})^{\omega}$, the* infinite *sequence $(g(\epsilon) \cup Y_0), \{g(Y_0 \cdots Y_i) \cup Y_{i+1}\}_{i \geq 0}$ does* not *have a finite prefix that satisfies φ.*

The function g is a *certificate* of unrealizability, as it proves that the environment has a strategy to prevent the agent from realizing φ. Theorem 1 characterizes the existence of certificates of unrealizability for an $\mathsf{LTL_f}$ specification in terms of existence of winning strategies, and follows directly from Definitions 1 and 2.

Theorem 1. *An $\mathsf{LTL_f}$ specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is realizable iff no certificate of unrealizability for $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ exists.*

Existing tools for $\mathsf{LTL_f}$ realizability and synthesis do not provide a certificate or explanation as to why a problem *cannot* be solved. Having such a certificate can be crucial to understanding or debugging errors in the modeling of the $\mathsf{LTL_f}$ problem, and allows for independent verification that the problem has no solution. A certificate is an important

tool in the arsenal of those who design and verify specifications and has a myriad of different applications in business process design and modeling.

5 Approach

Given an $\mathsf{LTL_f}$ specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, our approach to $\mathsf{LTL_f}$ synthesis and unrealizability comprises four steps:

(1) Preprocess $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ to eliminate variables from $\mathcal{X}$ and $\mathcal{Y}$ that do not appear in $\mathsf{LTL_f}$ formula, φ.

(2) Transform φ into NFA.

(3) Encode $\mathcal{X}, \mathcal{Y}$ together with the NFA states and dynamics from (2) as a FOND planning problem instance.

(4) Find a solution to the FOND problem constructed in (3).

The following sections describe the algorithmic details of a complete mapping from $\mathsf{LTL_f}$ synthesis to strong planning (Section 7), and from $\mathsf{LTL_f}$ unrealizability to strong cyclic planning (Section 8).

Solving $\mathsf{LTL_f}$ Realizability and Unrealizability The duality between $\mathsf{LTL_f}$ synthesis and the existence of certificates of unrealizability (Theorem 1) can be exploited to determine whether an specification is realizable or otherwise unrealizable. Corollaries 1 and 2 formalize these conditions. As such, realizability and unrealizability of a given $\mathsf{LTL_f}$ specification can be determined by running two complete solvers for $\mathsf{LTL_f}$ synthesis and unrealizability certificates computation, respectively.

Corollary 1. *An $\mathsf{LTL_f}$ specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is realizable when either $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ has a winning strategy, or no certificates of unrealizability for $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ exists.*

Corollary 2. *An $\mathsf{LTL_f}$ specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ is unrealizable when either $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ has a certificate of unrealizability, or no winning strategy for $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ exists.*

6 LTL-f Synthesis and Planning

Our objective in this section is to establish a clear mapping between $\mathsf{LTL_f}$ synthesis and FOND planning, two models for sequential decision-making. Other attempts to map synthesis and planning have been done to cast FOND planning as a reactive synthesis task (e.g. (Sardiña and D'Ippolito 2015)). The focus of this paper is in the other direction.

6.1 FOND Planning as LTL-f Synthesis

This section describes a polynomial reduction of a FOND problem $\mathcal{P} = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ into an $\mathsf{LTL_f}$ synthesis problem $\langle \mathcal{X}_\mathcal{P}, \mathcal{Y}_\mathcal{P}, \varphi_\mathcal{P} \rangle$. Inspired by encodings of planning into SAT (Rintanen, Heljanko, and Niemelä 2006), our reduction generates a single $\mathsf{LTL_f}$ formula $\varphi_\mathcal{P} \coloneqq \varphi_{init} \rightarrow \varphi_{env} \rightarrow (\varphi_{agt} \wedge \varphi_g)$, where φ_{init} models the initial state $\mathcal{I}$, φ_{env} and φ_{agt} model the dynamics of $\mathcal{P}$, and φ_g models the goal $\mathcal{G}$.

For each fluent $f \in \mathcal{F}$ and each action $a \in \mathcal{A}$ we create a variable in $\mathcal{Y}_\mathcal{P}$. $\mathcal{X}_\mathcal{P}$ is defined as $\{x_0, \ldots, x_M\}$, where $M = \lfloor \log(\max_{a \in \mathcal{A}} |\mathit{Eff}_a|) \rfloor$. An assignment σ to all variables in $\mathcal{X}_\mathcal{P}$ can be seen as encoding the number $\sum_{i=0}^{M} 2^i [\sigma(x_i) = true]$; intuitively, once an action is executed by the agent, this number is used to encode which of its effects triggers. Below, we denote as Γ_k the formula of variables in $\mathcal{X}_\mathcal{P}$ that encodes number k.

Now we describe each part of $\varphi_\mathcal{P}$. Formula φ_{agt}, which models the agent's action choice is a conjunction of two formulae that must hold globally. The first formula models the action preconditions and corresponds to the conjunction of formulae of the form $(a \rightarrow \bigwedge_{\ell \in Pre_a} \ell)$, for every $a \in \mathcal{A}$. The second formula establishes that exactly one action is performed at any time, by conjoining $\bigvee_{a \in \mathcal{A}} a$, and $\bigwedge_{a,a' \in \mathcal{A}, a \neq a'} (a \rightarrow \neg a')$.

Formula φ_{env} describes how the environment reacts to the execution of actions. Assuming that $\mathit{Eff}_a = \{e_0, \ldots, e_n\}$, formula χ_a defines the successor state, which depends on the setting of the $\mathcal{Y}$ variables by the environment. Intuitively, if the environment made Γ_k true, then the k-th effect of a triggers. The last effect e_n triggers if none of the the preceding effects triggers. Formally $\chi_a = (\bigwedge_{i=0}^{n-1} \neg \Gamma_i) \rightarrow (\bigcirc f \equiv \phi_{f,e_n}) \wedge \bigwedge_{i=0}^{n-1} (\Gamma_i \rightarrow (\bigcirc f \equiv \phi_{f,e_i}))$, where $\phi_{f,e}$ is a propositional formula that encodes all conditions under which f is true in the next state after outcome e of action a occurs. Finally, φ_{env} conjoins formulae $\Box(a \rightarrow \bullet\bot \vee \chi_a)$ for each action $a \in \mathcal{A}$, $\varphi_{init} \coloneqq \bigwedge_{\mathcal{I} \models l} l$ is the conjunction of the literals in the initial state $\mathcal{I}$, and $\varphi_g \coloneqq \Diamond \mathcal{G}$.

Theorem 2. *A FOND problem $\mathcal{P} = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ has a strong solution iff $\langle \mathcal{X}_\mathcal{P}, \mathcal{Y}_\mathcal{P}, \varphi_\mathcal{P} \rangle$ is realizable.*

Proof sketch. The construction is such that, if $\langle \mathcal{X}_\mathcal{P}, \mathcal{Y}_\mathcal{P}, \varphi_\mathcal{P} \rangle$ is realizable, then a winning strategy defines a policy that is a strong solution for $\mathcal{P}$. In the other direction, a winning strategy for the agent can be constructed from unfolding a strong policy for the FOND problem. □

Theorem 3. *FOND strong planning can be reduced to $\mathsf{LTL_f}$ synthesis in polynomial time*

6.2 LTL-f Synthesis as FOND Planning

In this section we show a reduction of $\mathsf{LTL_f}$ synthesis into FOND planning. The reduction of an $\mathsf{LTL_f}$ specification φ constructs a FOND planning problem $\mathcal{P}_\varphi = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ that simulates the dynamics of the 2-player game between the environment and the agent over an NFA $A_\varphi = \langle Q, \Sigma, q_0, \delta, \alpha \rangle$ that accepts the models of φ. The construction of A_φ is worst-case exponential, and the components of $\mathcal{P}_\varphi$ are polynomial in the size of A_φ. Hence, the reduction is exponential.

The components of $\mathcal{P}_\varphi$ are defined as follows. The set of fluents is $\mathcal{F} \coloneqq \mathcal{X} \cup Q \cup \{\mathtt{env}, \mathtt{agt}, \mathtt{goal}\}$. The initial state $\mathcal{I} \coloneqq \{\mathtt{env}, q_0\}$ sets the turn to the environment player, and the initial state of the automaton. Since the environment moves are uncontrollable to the agent, these are simulated with a non-deterministic action $a_{\mathtt{env}}$ with precondition $\{\mathtt{env}\}$ and non-deterministic outcomes $\mathit{Eff}_{a_{\mathtt{env}}} = \big\{\{\mathtt{agt}, \neg\mathtt{env}\} \cup X \mid X \in 2^\mathcal{X}\big\}$. The moves of the agent player are controllable, and are thus simulated with deterministic actions in the set $\mathcal{A}_{\mathtt{agt}} = \big\{a_Z \mid Z \in 2^\mathcal{X} \times 2^\mathcal{Y}\big\}$. Action a_Z, where $Z = (X, Y)$ simulates the agent playing Y after the environment has played X. The precondition of a_Z is $\{\mathtt{agt}\} \cup X$. The effects of a_Z update the automaton configuration according to δ and the input $X \cup Y$, and

reestablish the turn of the environment using conditional effects. Specifically, $a_{(X,Y)}$ has conditional effect $q \to q'$ for every q, q' such that $\delta(q, X \cup Y) = q'$, it has conditional effect $q \to \texttt{goal}$ for every $q \in Q$ such that $\delta(q, X \cup Y)$ is a final state, it has conditional effect $q \to \neg q$ for every $q \in Q$, and finally it has unconditional effects that remove $\texttt{agt}$ and add $\texttt{env}$. The set of planning actions is $\mathcal{A} := \{a_{\texttt{env}}\} \cup \mathcal{A}_{\texttt{agt}}$. Finally, the goal condition is $\mathcal{G} = \{\texttt{goal}\}$.

The actions of plans in the compiled FOND problem alternate $a_{\texttt{env}}$ with an action in $\mathcal{A}_{\texttt{agt}}$. Execution of an action sequence $a_{\texttt{env}}, a_{Z_0}, \ldots, a_{Z_m}$ simulates played turns $w = (X_0 \cup Y_0) \cdots (X_m \cup Y_m)$, and yields a state s_m that entails some automaton configuration $Q_m \in 2^Q$. The dynamics of the compiled FOND problem enforces the following property: $q \in Q_m$ iff there exists a run of A_φ on w that finishes in q. In other words, Q_m captures *all* the runs of the automaton. As such, the goal condition $\texttt{goal}$ is reached when the simulated play satisfies φ.

A winning strategy for $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ can be constructed from a strong policy π that solves $\mathcal{P}_\varphi$. Intuitively, $f(X_0 \cdots X_n)$ is obtained by unfolding π in $\mathcal{P}_\varphi$, and selecting the environment action effects that follows the sequence $X_0 \cdots X_n$. Such strategy can be implemented compactly as a finite-state controller (FSC).

Theorem 4. *An* $\mathsf{LTL_f}$ *synthesis problem* $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ *is realizable iff the FOND problem* $\mathcal{P}_\varphi$ *has a strong solution.*

Theorem 5. $\mathsf{LTL_f}$ *synthesis can be reduced into strong FOND planning in worst-case exponential time.*

7 LTL-f Synthesis via Planning

In Section 6 we established a mapping between $\mathsf{LTL_f}$ synthesis and FOND planning. In this section we describe the algorithmic details of a reduction from $\mathsf{LTL_f}$ synthesis to FOND planning, corresponding to step number **(3)** in our approach described in Section 5. In contrast to the mappings described in Section 6, we focus on performance and introduce a number of effective techniques to exploit structure. We use a STRIPS-like propositional language that can be adapted for use with standard FOND planners that use PDDL or SAS+ description model languages (e.g. myND (Mattmüller et al. 2010), PRP (Muise, McIlraith, and Beck 2012)). If desirable, more compact compilations can be easily obtained by using numeric fluents and conditional effects.

We assume that an NFA $A_\varphi = \langle Q, \Sigma, q_0, \delta, \alpha \rangle$ for $\mathsf{LTL_f}$ specification formula φ is given, and construct a FOND problem $\mathcal{P}_\varphi = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$. Below, for each pair $q, q' \in Q$, $\mathrm{guard}(q, q')$ is a DNF formula equivalent to $\bigwedge_{(q,\psi,q') \in \delta} \psi$. The set T of transitions is defined as $\{(q, \mathrm{guard}(q, q'), q') \mid q, q' \in Q\}$. We denote by $\mathrm{macro}(t)$, for $t = (q, \psi, q')$, the set of transitions in T of the form (q, ψ', q').

The dynamics of $\mathcal{P}_\varphi$ simulates the two-player game between the environment and agent players. Each round is simulated with three consecutive stages. The first stage simulates, non-deterministically, the move of the environment. The second stage simulates the move of the agent. The third mode synchronizes the states of the automaton by performing those in accordance with the input move being simulated.

$$
\begin{aligned}
\mathcal{F} &:= \{\texttt{autState(q)}\}_{q \in Q} \cup \{\texttt{poss(t)}\}_{t \in T} \cup \{\texttt{sync}\} \\
&\quad \cup \{\texttt{isTurn}(\texttt{v}_{|\mathcal{X}|})\} \cup \{\texttt{isTurn(v)}\}_{v \in \mathcal{X}} \cup \mathit{Facts} \\
\mathit{Facts} &:= \{\texttt{isAcc(t)}\}_{t.dest \in \alpha} \\
\mathcal{I} &:= \{\texttt{sync}, \texttt{autState}(\texttt{q}_0)\} \\
\mathcal{A} &:= \{\texttt{continue}\} \cup \{\texttt{playX}(v)\}_{v \in \mathcal{X}} \\
&\quad \cup \{\texttt{transAcc(t)}, \texttt{transRej(t)}\}_{t \in T} \\
&\quad \cup \{\texttt{startSync}\} \cup \{\texttt{syncAutState(q)}\}_{q \in Q} \\
\mathcal{G} &:= \{\texttt{winning}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Pre}_{\texttt{playX}(v_i)} &:= \{\texttt{isTurn}(\texttt{v}_\texttt{i})\} \\
\mathit{Eff}_{\texttt{playX}(v_i)} &:= \{\texttt{isTurn}(\texttt{v}_{\texttt{i+1}}), \neg\texttt{isTurn}(\texttt{v}_\texttt{i})\} \\
&\quad \cup \mathit{oneof}(e_1, e_2) \\
e_1 &:= \{\neg\texttt{poss(t)}\}_{\neg v_i \in \mathrm{Lits}(\mathrm{guard}(t))} \\
e_2 &:= \{\neg\texttt{poss(t)}\}_{v_i \in \mathrm{Lits}(\mathrm{guard}(t))} \\
\mathit{Pre}_{\texttt{transAcc(t)}} &:= \{\texttt{isTurn}(\texttt{v}_{|\mathcal{X}|}), \texttt{poss(t)}, \texttt{isAcc(t)}\} \\
\mathit{Eff}_{\texttt{transAcc(t)}} &:= \{\texttt{autState}(\texttt{q}'), \neg\texttt{poss(t)}, \texttt{winning}\} \\
&\quad \cup \{\neg\texttt{poss}(\texttt{t}')\}_{t' \in \mathrm{mutex}(t) \cup \mathrm{macro}(t)} \\
\mathit{Pre}_{\texttt{transRej(t)}} &:= \{\texttt{isTurn}(\texttt{v}_{|\mathcal{X}|}), \texttt{poss(t)}, \neg\texttt{isAcc(t)}\} \\
\mathit{Eff}_{\texttt{transRej(t)}} &:= \{\texttt{autState}(\texttt{q}'), \neg\texttt{poss(t)}\} \\
&\quad \cup \{\neg\texttt{poss}(\texttt{t}')\}_{t' \in \mathrm{mutex}(t) \cup \mathrm{macro}(t)} \\
\mathit{Pre}_{\texttt{startSync}} &:= \{\texttt{isTurn}(\texttt{v}_{|\mathcal{X}|})\} \\
\mathit{Eff}_{\texttt{startSync}} &:= \{\texttt{sync}, \neg\texttt{isTurn}(\texttt{v}_{|\mathcal{X}|})\} \\
&\quad \cup \{\neg\texttt{poss(t)}\}_{t \in T} \\
\mathit{Pre}_{\texttt{syncAutState(q)}} &:= \{\texttt{sync}, \texttt{autState(q)}\} \\
\mathit{Eff}_{\texttt{syncAutState(q)}} &:= \{\texttt{poss(t)}\}_{t \in T, t.orig = q} \\
&\quad \cup \{\neg\texttt{autState(q)}\} \\
\mathit{Pre}_{\texttt{continue}} &:= \{\texttt{sync}\} \cup \{\neg\texttt{autState(q)}\}_{q \in Q} \\
\mathit{Eff}_{\texttt{continue}} &:= \{\texttt{isTurn}(\texttt{v}_0), \neg\texttt{sync}\}
\end{aligned}
$$

Figure 1: FOND compilation $\mathcal{P}_\varphi := \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ for an $\mathsf{LTL_f}$ synthesis problem $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ and given NFA A_φ.

In the first stage, the move of the environment is simulated with a cascade of actions $\texttt{playX}(v_i)$ for each uncontrollable variable $v_i \in \mathcal{X}$, indexed for $0 \leq i < |\mathcal{X}|$. These actions have non-deterministic effects that simulate an uncontrollable assignment to variables in $\mathcal{X}$. For convenience, we include an auxiliary fluent $v_{|\mathcal{X}|}$.

In the second mode, the assignment to variables in $\mathcal{Y}$ is simulated implicitly by means of *transition actions* $\texttt{transAcc(t)}$ and $\texttt{transRej(t)}$, one for each $t \in T$. Intuitively, the agent decides which automaton transitions perform within the ones that are feasible. Then, the dynamics of the problem simulates the assignment to variables $\mathcal{Y}$ necessary to satisfy $\mathrm{guard}(t)$, and deems unfeasible all transitions t' whose guard is mutually exclusive with such assignment. Feasible transitions $t \in T$ are identified by the

truth of fluents $\texttt{poss}(\texttt{t})$. More formally, $\texttt{transAcc}(\texttt{t})$ (resp. $\texttt{transRej}(\texttt{t})$) sets $\texttt{poss}(\texttt{t}')$ false to all $t' \in \text{mutex}(t)$, where $t' \in \text{mutex}(t)$ if $\text{guard}(t) \wedge \text{guard}(t') \models \bot$. For convenience, transition actions can deem unfeasible all transitions $t' \in \text{macro}(t)$, as it can be proved that $\texttt{transAcc}(\texttt{t}')$ and $\texttt{transRej}(\texttt{t}')$ are no longer relevant actions. Transition actions for $t = (q, \psi, q') \in T$ turn the fluent $\texttt{autState}(\texttt{q}')$ true to acknowledge that there exists a run of A_φ on the word w being simulated that finishes in q'. When q' is an accepting state (indicated by $\texttt{t.dest} \in \alpha$), action $\texttt{transAcc}(\texttt{t})$ turns the fluent `winning` true to acknowledge that w is accepting.

The third mode starts with the action `startSync`. In the third stage, actions $\texttt{syncAutState}(\texttt{q})$ synchronize the automaton states, and deem feasible all transitions $t \in T$ that have origin state q (indicated by $\texttt{t.orig} = \texttt{q}$). At the end of the third stage, action `continue` initiates the simulation of a new round by reestablishing the dynamics to the first stage.

To conclude the description of $\mathcal{P}_\varphi$, the initial state $\mathcal{I}$ starts in synchronization mode, with the initial automaton state fluent $\texttt{autState}(\texttt{q}_0)$ set true. The goal condition is to set the fluent `winning` true.

Theorem 6 states the correctness of the compilation. A strategy $f : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ that realizes φ can be constructed from a strong solution to $\mathcal{P}_\varphi$ by unfolding the policy over the simulated turns of the two-player game. This can be implemented compactly in form of a finite-state controller (FSC). Because FOND is EXPTIME-complete (Rintanen 2004) and our translation is exponential in the size of the formula, the overall approach is worst-case 2-EXPTIME in the size of φ. This is because the size of the compilation is worst-case exponential in the size of A_φ, and therefore worst-case double exponential in the size of φ.

Theorem 6. *An* $\mathsf{LTL_f}$ *specification* $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ *is realizable iff the compiled FOND problem* $\mathcal{P}_\varphi$ *has a strong solution.*

Corollary 3. *A FSC that implements a solution to the synthesis problem for specification* $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ *can be constructed from a strong solution to the compiled FOND problem* $\mathcal{P}_\varphi$*.*

Theorem 7. *The approach to solve* $\mathsf{LTL_f}$ *synthesis via compilations to FOND planning is worst case double exponential in the size of the formula.*

We close this section with a discussion on the lazy determinization of the automaton and state abstractions performed inherently within the dynamics of the compilation.

Lazy Determinization and Powerset Construction The powerset construction is a well-known technique to determinize a NFA into a DFA (Rabin and Scott 1959). States in the DFA– so-called *macrostates* – are sets of states of the original NFA, and macrostate transitions are constructed by transitioning all NFA states in the macrostate. The powerset construction is worst-case exponential in the size of the NFA. Like in the powerset construction, planning states in our compilation maintain a set of NFA states. However, the determinization is *lazy*, meaning that macrostates are instantiated on the fly in planning time only when necessary. This leads to potentially exponential savings, when only a small subset of macrostates are relevant to the search process. Similar techniques to avoid instantiation of the deterministic automaton have been used in the LTL synthesis tool *Acacia*$^+$ (Bohy et al. 2012) and $\mathsf{LTL_f}$ synthesis tool *Syft* (Zhu et al. 2017).

Symbolic Determinization and State abstractions Unlike the powerset construction, the transition of a macrostate in our compilation is not obliged to apply *all* possible automaton state transitions. Instead, the agent *can* decide not to apply certain transitions when these are not *relevant*. Consequently, planning states can maintain more compact macrostates, that we call *partial macrostates* for its parallelism with so-called partial states in planning. The policy that applies for a certain partial macrostate is also valid for all macrostates that contain (or entail) the partial macrostate. As such, partial macrostates can be interpreted as *families of macrostates*, or *state abstractions* (e.g (Knoblock 1994)). State abstractions are possible thanks to the symbolic determinization of the NFA, that represents macrostates as sets of fluents instead of instantiating a new fluent for each macrostate.

8 Certificates of Unrealizability

A sound and complete procedure for determining the realizability of an $\mathsf{LTL_f}$ synthesis problem can naturally be used for instances that have no solution. However, by casting it as a FOND problem to be solved leaves us with little recourse to understand the source of inconsistency – generally speaking, FOND solvers do not provide a certificate or explanation as to why a problem *cannot* be solved.

Our approach to compute a certificate of unrealizability compiles the synthesis problem for $\mathsf{LTL_f}$ specification φ into an instance of FOND planning $\overline{\mathcal{P}}_\varphi$ that simulates the two-player game between the environment and agent players. The details of the compilation differs from the construction of $\mathcal{P}_\varphi$ presented in Section 7 in four key aspects. First, in this setting the moves of the environment are controllable, and simulated with deterministic actions `setX` and `unsetX`. The moves of the agent are uncontrollable, and are simulated with non-deterministic actions `playY`. Second, $\overline{\mathcal{P}}_\varphi$ integrates the dynamics of an NFA automaton $A_{\neg\varphi}$ that accepts the models of $\neg\varphi$. Like in the compilation presented in Section 7, the dynamics of $\overline{\mathcal{P}}_\varphi$ has the property that for a simulated sequence of moves $w = (X_0 Y_0) \cdots (X_n Y_n)$, the fluent `winning` can be made true to acknowledge that there exists a run of $A_{\neg\varphi}$ on w that finishes in an accepting state, and therefore satisfies $\neg\varphi$ (Lemma 1). Third, the action `continue` is non-deterministic, and applicable if the fluent `winning` holds; otherwise, a deadend is reached. The extra effect reestablishes the turn of the environment, and allows for simulation of plans of infinite length. Finally, the initial state is $\mathcal{I} \coloneqq \{\texttt{winning}, \texttt{sync}, \texttt{autState}(\texttt{q}_0)\}$, and the goal is $\mathcal{G} \coloneqq \{\texttt{goal}\}$. Theorem 8 states the correspondence between strong cyclic solutions to $\overline{\mathcal{P}}_\varphi$ and the unrealizability of $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$. A certificate of unrealizability can be obtained by unfolding a strong cyclic solution (Theorem 4).

Finally, the overall approach to obtain certificates of unrealizability is worst-case double exponential in the size of φ. The proof follows a similar argument than the one used

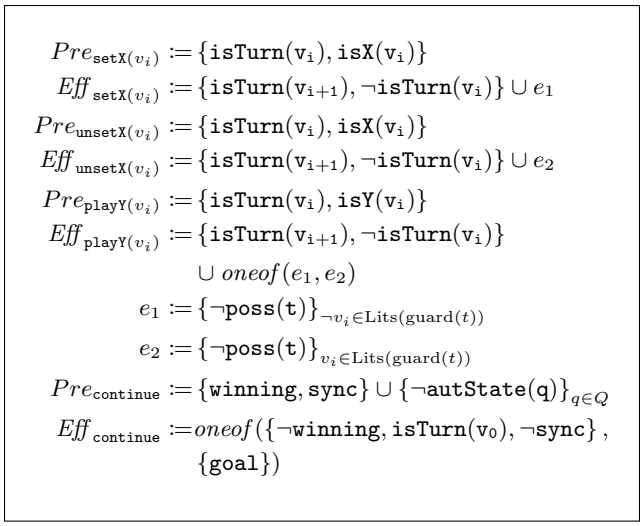


Figure 2: Components of the FOND compilation to obtain certificates of unrealizability that differ from the compilation described in Figure 1.

in Section 7 to establish exponential bounds on the size of the problem with respect to the size of the NFA.

Lemma 1. *A sequence of moves $w = (X_0Y_0)\cdots(X_nY_n)$ satisfies $\neg\varphi$ iff there exists a plan in $\overline{\mathcal{P}}_\varphi$ that simulates w and makes fluent* `winning` *true.*

Theorem 8. *An $\mathsf{LTL_f}$ synthesis problem $\langle\mathcal{X},\mathcal{Y},\varphi\rangle$ is unrealizable iff the compiled FOND problem $\overline{\mathcal{P}}_\varphi$ has a strong cyclic solution.*

Proof. By definition, a policy π is a strong cyclic solution iff the goal is reachable (by following π) from all states reachable by π. In $\overline{\mathcal{P}}_\varphi$ this happens iff executions never reach a deadend and action `continue` is always applicable at the end of the third mode. This happens iff the fluent `winning` can be made true at the end of each simulated round. Lemma 1 completes the proof. □

Corollary 4. *A certificate of unrealizability for an $\mathsf{LTL_f}$ synthesis problem $\langle\mathcal{X},\mathcal{Y},\varphi\rangle$ can be obtained from strong cyclic solutions to $\overline{\mathcal{P}}_\varphi$.*

Theorem 9. *The NFA-based reductions of $\mathsf{LTL_f}$ synthesis to check unrealizability are in double exponential time on the size of the formula.*

9 Leveraging Multiple Automata

Recall that the major source of complexity for LTL and $\mathsf{LTL_f}$ synthesis is the automaton construction. To improve scalability, our approach leverages a technique that decomposes the original $\mathsf{LTL_f}$ formula into multiple subformulae and then constructs multiple (smaller) automata, rather than a single larger automaton. To do so, $\mathsf{LTL_f}$ formula φ is first converted to Negation Normal Form (NNF) and then selectively decomposed into conjunctive subformulae, φ_i, balancing the number and relative size of subformulae. Next, each subformula, φ_i, is transformed into an NFA, A_i, that accepts the models of φ_i.

To define an appropriate acceptance criterion for the aggregate of the automata, we replace each φ_i in the NNF of φ with a new distinguished variable acc_i that is true iff A_i accepts w. The resulting formula defines the acceptance criterion for the aggregate of our A_i's. Baier and McIlraith (2006) proved that this acceptance criteria captures all and only the models of φ.

Leveraging these multiple automata in the approaches of previous sections is straightforward. The FOND compilations described in Sections 7 and 8 are modified to reflect the dynamics of all the individual automata. The unique fluent `winning`, is replaced by a set of fluents, `winning`(i), one for each automaton A_i. Intuitively, these fluents play the role of acc_i above and are made true precisely when the simulated input word is accepting for A_i, and false otherwise. The goal of the FOND problem is then defined by replacing each φ_i in the NNF of φ with the fluent `winning`(i).

From the correctness of our FOND encodings and the correctness of the acceptance criterion for the collection of automata, it is easy to see that the dynamics of the FOND problem capture all and only the models of φ. With these modifications, our automata-based compilations are correct.

10 Empirical Evaluation

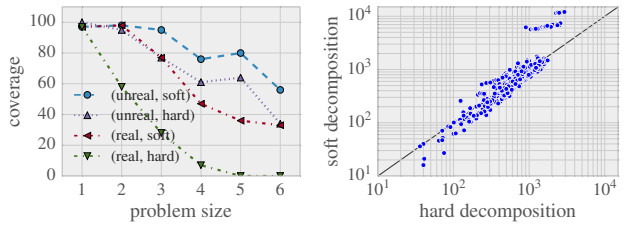
We implemented our proposed approaches in a tool we have named *SynKit*. *SynKit* is able to generate strategies for realizable synthesis problems, to determine whether a problem is realizable or unrealizable, and to construct a certificate if it is unrealizable. We evaluated the performance of *SynKit* in comparison to state-of-the-art tool *Syft* (Zhu et al. 2017). All tests were conducted in an Intel Xeon E5-2430 2.2GHz machine, with processes limited to 4GB of memory and 30 minutes runtime.

Off-the-Shelf Components We employed *Syft*'s code to translate $\mathsf{LTL_f}$ formulae into first-order logic (FOL), and software system *Mona* (Henriksen et al. 1995) to translate FOL formulae into NFA(**). We used the planner myND (Mattm¨uller et al. 2010) to compute strong and strong cyclic solutions to the compiled FOND problems.

Benchmarks We conducted experiments with a set of 24 benchmark problems from the *lilydemo* benchmark set (Jobstmann and Bloem 2006), drawn from previous Syntcomp LTL synthesis competitions (over infinite traces). These problems are easy to solve when the specification formulae are interpreted over finite traces. Consequently, we constructed new problems by conjoining a number of *lilydemo* benchmarks, that from now on we call *subproblems*. More formally, the new problems have the form $\langle\mathcal{X},\mathcal{Y},\varphi\rangle$, where $\mathcal{X} = \bigcup\mathcal{X}_i$, $\mathcal{Y} = \bigcup\mathcal{Y}_i$, and $\varphi = \bigwedge\varphi_i$ for certain subproblems $\langle\mathcal{X}_i,\mathcal{Y}_i,\varphi_i\rangle$. We refer to the number of subproblems that originate the specification formula as the *size* of a problem. We generated, at random, 100 problems for each problem size ranging from 1 to 6.

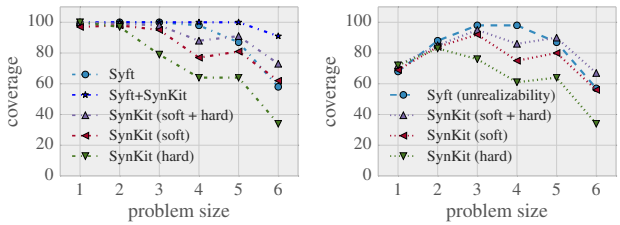
Configurations *SynKit* can be configured to search for winning strategies or certificates of unrealizability. As explained in Sections 7 and 8, each objective produces different FOND

(**) Correction: the automata produced by Mona were DFAs.



(a) $\mathsf{LTL_f}$ Realizability (b) FOND Problem Size

Figure 3: Performance of *SynKit* configured to compute winning strategies (real) and certificates of unrealizability (unreal). FOND compilations were generated using *soft* and *hard* multiple automata decompositions.



(a) $\mathsf{LTL_f}$ Realizability (b) Certificates Generation

Figure 4: Number of problems solved by different configurations of *SynKit* for $\mathsf{LTL_f}$ realizability (Figure 4a) and unrealizability certificates generation (Figure 4b). The performance is evaluated relative to *Syft*.

compilations. Additionally, *SynKit* can make use of multiple automata decompositions of the specification formula. In our experiments, we made use of two types of automata decompositions, that we name *soft* and *hard*. Soft decompositions transform the $\mathsf{LTL_f}$ formula of each subproblem into an NFA. For $\mathsf{LTL_f}$ synthesis, hard decompositions consider the CNF representation of the $\mathsf{LTL_f}$ specification formula, and transform each clause into an NFA. For $\mathsf{LTL_f}$ unrealizability, hard decompositions consider the DNF representation of the negation of the $\mathsf{LTL_f}$ specification formula, and transform each clause into an NFA.

10.1 Multiple Automata Decompositions

We conducted experiments to evaluate the scalability of the system with different automata decompositions. The first thing to note is that the use of multiple automata decompositions is necessary to scale to large problems. Indeed, problems with size greater than 1 have $\mathsf{LTL_f}$ specification formulae that frequently transformed into large automata and generated FOND problems of prohibitively large size. In more general terms, we observed that some large formulae could not be transformed into automata by the approach taken by *Syft*, as it ran into memory issues. This problem never occurred with our decompositions, which decompose the formula into smaller subformulae that are easier to transform.

We tested four different configurations of *SynKit*: using *soft* and *hard* automata decompositions, and searching for winning strategies and certificates of unrealizability. Figure 3a shows the coverage of the realizability tests performed by each of these configurations. Recall that the realizability can be determined by finding a solution (i.e. a winning strategy, or a certificate), or proving that none exists.

Clearly, the configuration that performed soft decompositions, and searched for certificates dominated the others. On the other extreme, the configuration that performed hard decompositions, and searched for winning strategies manifested the worst performance. This dominance could be related to the fact that most of the problems being tested were unrealizable. Notably, the configurations that performed soft decompositions dominated the configurations that performed hard decompositions. Thus, it seems that automata decompositions are necessary to scale, but decomposing the $\mathsf{LTL_f}$ formula into too many automata may be counterproductive and affect search performance. Soft and hard decompositions resulted in FOND problems of similar size, with a few exceptions where soft decompositions resulted in FOND problems of size one order of magnitude higher (Figure 3b). The discrepancy occurs because a reduced number of subproblems have specification formula that transforms – via soft decompositions – into significantly large automata. In these cases, hard decompositions can be advantageous and generate more tractable FOND problems.

10.2 Global Performance

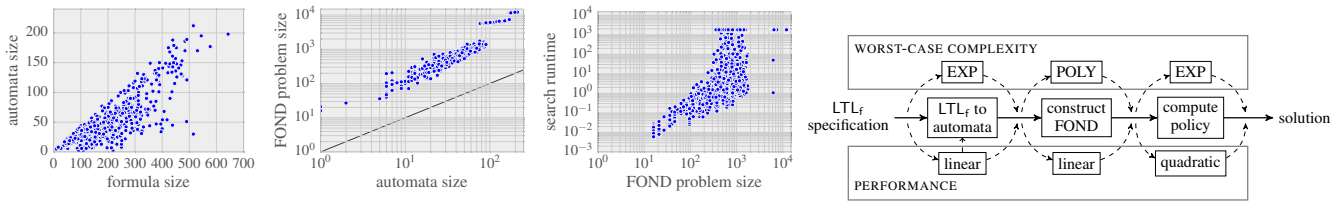
We conducted experiments to evaluate the performance of our approaches to $\mathsf{LTL_f}$ realizability, synthesis, and certificate generation. We detail the results of our tests below.

$\mathsf{LTL_f}$ **Synthesis** All realizable $\mathsf{LTL_f}$ specifications in our problem set have sizes 1 or 2. Even though automata decompositions were needed to scale, *SynKit* was able to find winning strategies to realizable $\mathsf{LTL_f}$ specifications with either soft or hard decompositions of the formula.

$\mathsf{LTL_f}$ **Realizability** $\mathsf{LTL_f}$ realizability can be determined more efficiently by running two configurations of *SynKit* in parallel: one that searches for winning strategies, and another one that searches for unrealizability certificates. $\mathsf{LTL_f}$ realizability can be determined when either a winning strategy is found, a certificate is found, or one of the searches concludes that no solution exists. We refer to these parallel searches as *portfolios*.

We evaluated the performance of portfolios that used soft (*SynKit (soft)*) and hard (*SynKit (hard)*) decompositions, and also a portfolio that combined both methods above (*SynKit (soft + hard)*). *Syft* had better coverage than *SynKit (soft)* and *SynKit (hard)* alone, especially in problems with large size. Interestingly, these configurations of *SynKit* solved different sets of problems, and the portfolio *SynKit (soft + hard)* altogether demonstrated better coverage than *Syft* in problems of large size. Still, none of these methods strictly dominated each other. A final round of experiments confirmed that there is value in combining *Syft* with all configurations of *SynKit* – in a portfolio that we denote by *Syft + SynKit* – resulting in a configuration with the best coverage.

Certificates of Unrealizability *SynKit* is the only existing



(a) $\mathsf{LTL_f}$ to Automata (b) FOND Problem Size (c) Search Runtime (d) Complexity vs. Performance

Figure 5: Observed performance in each phase of *SynKit*. Automata constructions with *soft* decompositions are linear in the size of the formulae (Figure 5a). The compiled FOND problems are linear in the size of the multiple automata (Figure 5b). Finally, the search runtime for most problems grows quadratically with the size of the FOND problem (Figure 5c).

tool that computes certificates of unrealizability. For reference, we compared the number of unrealizability certificates computed by *SynKit* with the number of problems that were reportedly unrealizable (i.e. no solution for the synthesis problem exists) by *Syft*. Figure 4b summarizes the results. *SynKit (hard)* was able to find unrealizability certificates to nearly all problems of sizes 1, 2, and 3 that are unrealizable. Its performance suffered in problems of larger size, but the coverage approximated that of *Syft* in problems with size 5 and 6. The best coverage was achieved with the portfolio *SynKit (soft + hard)*, especially in problems of largest size, where it manifested better coverage than *Syft* – with the advantage of computing certificates of unrealizability.

10.3 Empirical Difficulty

We studied the complexity that the different phases of our approach manifested in practice, with the objective to contrast it with the worst-case hardness. We used the configuration of *SynKit* that performs *soft* decompositions, and searches for certificates of unrealizability. The results are summarized in Figure 5d, and discussed below.

$\mathsf{LTL_f}$ to Automata Decompositions of the $\mathsf{LTL_f}$ formulae into multiple automata allow for effective automata transformations, with almost instantaneous run times that were always lower than 8 seconds per problem. The sizes of the multiple automata, measured as the sum of the sizes of all automatons in the automata, did not result in the worst-case exponential explosion (Figure 5a). Rather, they seemed to grow linearly with the size of the formula – measured as the number of logical connectives and temporal operators.

Automata to FOND The general trend observed in our experiments shows that the size of the compiled FOND problems (measured as the sum of variables and ground actions) grows linearly with respect to the size of the multiple automata decompositions (Figure 5b). Automata decompositions for some problems resulted in significantly large automata, a phenomena that explains the clusters that we can also observe in Figure 3b. The number of actions in the FOND compilations that model automaton transitions is, worst-case, quadratic in the number of automaton states and exponential in the number of variables in the specification. In practice, we did not see such a combinatorial explosion.

Policy Search Figure 5c shows the distribution of planning search runtime with respect to the size of the problem. Time violations of 30 minutes are also represented. While the data is reasonably sparse, linear regression nonetheless indicates that the search runtime grows quadratically with the problem size, at least locally in problems of size lower than 500. As the problem size approached 10^3, the trend exhibited a more exponential behaviour. Even still, most of the problems could be handled by the FOND planner myND. Our experiments used myND as a unified software to compute strong and strong-cyclic solutions, and the runtimes obtained are not competitive with *Syft* – which obtained solutions in the order of seconds. However, more efficient alternative planners exist. In particular, PRP (Muise, McIlraith, and Beck 2012) is, in general, more efficient than myND in computing strong-cyclic solutions, and FIP (Fu et al. 2016) is a competitive strong planner that has been more recently presented.

11 Summary and Concluding Remarks

Synthesizing software from LTL specifications is a fundamental problem in computer science. Its finite trace variant, $\mathsf{LTL_f}$ synthesis, captures many interesting and important problems that have finite duration, including typical planning problems and problems involving business processes.

The contributions of this paper are theoretical, algorithmic, and experimental. In particular, we characterized the notion of a certificate of unrealizability for an $\mathsf{LTL_f}$ specification, and established its duality with $\mathsf{LTL_f}$ realizability. Next, we cast $\mathsf{LTL_f}$ synthesis and unrealizability as planning. From a theoretical perspective, we established the correspondence between $\mathsf{LTL_f}$ synthesis and FOND planning. From an algorithmic perspective, we constructed algorithms to find winning strategies and certificates of unrealizability. Our algorithms leverage a number of interesting techniques to exploit structure. Namely, state abstractions, lazy determinization of NFA, and the transformation of our $\mathsf{LTL_f}$ specification formula into multiple automata.

We implemented our algorithms in a synthesis tool that we named *SynKit*. *SynKit* is the first $\mathsf{LTL_f}$ synthesis tool that computes certificates of unrealizability. Our empirical evaluation illustrates that *SynKit* is a good complement to state-of-the-art $\mathsf{LTL_f}$ synthesis tool *Syft* in determining realizability and synthesis. Combined as a portfolio system *Syft* and *SynKit* represent the state of the art. Experiments highlighted some interesting computational properties that we plan to explore in future work.

Acknowledgements

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

References

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

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

Baier, J. A., and McIlraith, S. A. 2006. Planning with temporally extended goals using heuristic search. In *ICAPS*, 342–345.

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

Bohy, A.; Bruyère, V.; Filiot, E.; Jin, N.; and Raskin, J. 2012. Acacia+, a tool for LTL synthesis. In *CAV*, 652–657.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. A. 2017. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*.

Camacho, A.; Baier, J. A.; Muise, C. J.; and McIlraith, S. A. 2018. Synthesizing controllers: On the correspondence between LTL synthesis and non-deterministic planning. In *CCAI*. To appear.

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

Coles, A., and Coles, A. 2011. LPRPG-P: relaxed plan heuristics for planning with preferences. In *ICAPS*.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. In *IJCAI*, 1558–1564.

De Giacomo, G.; Maggi, F. M.; Marrella, A.; and Patrizi, F. 2017. On the disruptive effectiveness of automated planning for LTL$_f$-based trace alignment. In *AAAI*, 3555–3561.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *ICAPS*, 374–377.

Fu, J.; Jaramillo, A. C.; Ng, V.; Bastani, F. B.; and Yen, I. 2016. Fast strong planning for fully observable nondeterministic planning problems. *Annals of Mathematics and Artificial Intelligence* 78(2):131–155.

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

Henriksen, J.; Jensen, J.; Jørgensen, M.; Klarlund, N.; Paige, B.; Rauhe, T.; and Sandholm, A. 1995. Mona: Monadic second-order logic in practice. In *TACAS*.

Jobstmann, B., and Bloem, R. 2006. Optimizations for LTL synthesis. In *FMCAD*, 117–124.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern database heuristics for fully observable nondeterministic planning. In *ICAPS*, 105–112.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*, 172–180.

Patrizi, F.; Lipovetzky, N.; and Geffner, H. 2013. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *IJCAI*.

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages*, 179–190.

Pnueli, A. 1977. The temporal logic of programs. In *FOCS*, 46–57.

Rabin, M. O., and Scott, D. S. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development* 3(2):114–125.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Rintanen, J. 2004. Complexity of planning with partial observability. In *ICAPS*, 345–354.

Sardiña, S., and D'Ippolito, N. 2015. Towards fully observable non-deterministic planning as assumption-based automatic synthesis. In *IJCAI*, 3200–3206.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017. Symbolic LTLf synthesis. In *IJCAI*, 1362–1369.