Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

LTL Realizability via Safety and Reachability Games

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

Abstract

In this paper, we address the problem of LTL realizability and synthesis. State of the art techniques rely on so-called *bounded synthesis* methods, which reduce the problem to a *safety* game. Realizability is determined by solving synthesis in a *dual* game. We provide a unified view of duality, and introduce novel *bounded realizability* methods via reductions to *reachability* games. Further, we introduce algorithms, based on AI automated planning, to solve these safety and reachability games. This is the the first complete approach to LTL realizability and synthesis via automated planning. Experiments illustrate that reductions to reachability games are an alternative to reductions to safety games, and show that planning can be a competitive approach to LTL realizability and synthesis.

1 Introduction

LTL synthesis aims to compute a strategy that satisfies a Linear Temporal Logic (LTL) specification. The problem was first proposed in the context of reactive synthesis, and is central to the automated construction of controllers and certain classes of programs [Pnueli and Rosner, 1989].

LTL realizability and synthesis are commonly addressed as a two-player game between an agent and the environment, played over automata transformations of the LTL specification (so-called *automata games*). Each player has a disjoint set of variables, and the objective is to synthesize a strategy for setting the agent's ("controllable") variables such that the LTL specification is guaranteed to be satisfied, no matter how the environment sets its ("uncontrollable") variables.

Traditional approaches to LTL realizability and synthesis assume that the agent plays first. Interestingly, in recent years there has been a surge in the development of modern LTL synthesis tools based on automata games, and whereas LTL synthesis tool Lily [Jobstmann and Bloem, 2006] adopts an agent-first play protocol, many of the most successful LTL synthesis tools such as Unbeast [Ehlers, 2010], Acacia, and Acacia$^+$ [Filiot *et al.*, 2009; Bohy *et al.*, 2012] adopt an inverted turn protocol, where the environment plays first. Indeed, this *inverted turn-taking* has become the standard used in different incarnations of SYNTCOMP, the annual reactive synthesis competition (e.g. [Jacobs *et al.*, 2017]). SYNTCOMP has promoted standardization and facilitated tool comparison with the introduction of the high-level LTL input language TLSF [Jacobs *et al.*, 2016]. Participating tools (including Acacia$^+$) are compliant with TLSF.

To describe the different semantics associated with turn-taking protocols, the notions of *Mealy-* and *Moore-type* semantics has been introduced (e.g. [Ehlers, 2011; Khalimov *et al.*, 2013]), where intuitively Mealy-type semantics correspond to settings where the environment plays first, and Moore-type semantics correspond to settings where the agent plays first. The names originate from Mealy and Moore machines. When a winning strategy exists for one of these semantics, the strategy has a finite representation and is typically conveyed in terms of Mealy (resp. Moore) machines.

Inverting the order of agent-environment turn-taking corresponds to playing a *dual* game. We aim to provide a unified view of duality results for LTL realizability and synthesis, and automata games. To this end, we review the connection between LTL synthesis and automata games, and formalize a duality result for existence of winning strategies in automata games that replicates the duality results for synthesis. We further investigate different reductions to automata games that exploit duality to determine realizability, and introduce novel techniques that exploit these correspondences.

We exploit mappings from LTL synthesis to games, together with duality results, to chronicle different techniques to solve LTL realizability and synthesis (Table 1). These techniques extend well-known approaches, and establish the connection between Mealy and Moore semantics for LTL realizability and synthesis, and games over *Universal k-coBüchi Word* (UkCW) automata and *Non-deterministic k-Büchi Word* (NkBW) automata. From an algorithmic perspective, our approaches to LTL realizability via reduction to NkBW games provide an alternative to existing techniques based on UkCW. Whereas the latter are *safety* games, the former are *reachability* games. Finally, we address the realizability problem in

*This work was conducted while the author was affiliated with CSAIL, Massachusetts Institute of Technology.

safe and co-safe specifications, and show how these can be solved more efficiently.

We further present the first *complete* approach to LTL synthesis via Fully Observable Non-Deterministic (FOND) planning. Our approach constructs a planning problem that simulates an automata game, and that can be solved using state-of-the-art planners. While our contributions are intended to be largely theoretical, we conducted preliminary experiments to assess the practicality of the different techniques for LTL realizability and synthesis that we present. Preliminary results show that techniques employing NkBW automata can complement existing techniques based on UkCW automata, addressing realizability and synthesis problems that otherwise cannot be solved by existing tools. Moreover, they show that automated planning technology (and, in particular, FOND) can be an efficient approach to LTL realizability and synthesis. Finally, we believe that the NkBW game reductions presented here open the door to novel *bounded realizability* methods for LTL realizability and synthesis as reachability games, providing an alternative to bounded synthesis [Kupferman and Vardi, 2005; Schewe and Finkbeiner, 2007].

2 Background

2.1 Automata on Infinite Words

A *non-deterministic automaton* is a tuple $A = \langle Q, \Sigma, q_0, \delta, \alpha \rangle$, where Q is a finite set of automaton *states*, Σ is a finite alphabet of input letters, $q_0 \in Q$ is the *initial state* of the automaton, $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*, and $\alpha \subseteq Q$ is an *accepting condition*. We sometimes write $\delta(q, \sigma)$ to denote the set $Q' \subseteq Q$ such that $(q, \sigma, q') \in \delta$ for every $q' \in Q'$. Intuitively, an automaton A in state q can transition into any state in $\delta(q, \sigma)$ when it reads an input letter σ. When $|\delta(q, \sigma)| = 1$ for every $q \in Q$ and $\sigma \in \Sigma$, the automaton is *deterministic*.

Σ^ω denotes the set of all infinite words over Σ. A *run* of A on word $w = \sigma_1, \sigma_2, \ldots \in \Sigma^\omega$ is an infinite sequence of states $\rho = q_0, q_1, \ldots \in Q^\omega$ where $(q_{i-1}, \sigma_i, q_i) \in \delta$ for every $i > 0$. It is useful to define operators $\mathrm{occ}_\rho : Q \to \mathbb{N} \cup \{0, \infty\}$, indexed by runs ρ, that return for each $q \in Q$ the number of occurrences of q in ρ. In what follows, we review three types of acceptance conditions. A run ρ is accepting with the *Büchi* condition if $\mathrm{occ}_\rho(q) = \infty$ for some $q \in \alpha$ – that is, when some state in α occurs infinitely often in ρ. Conversely, a run ρ is accepting with the *coBüchi* condition if $\mathrm{occ}_\rho(q) < \infty$ for every $q \in \alpha$. Finally, ρ is accepting with the *k-Büchi* (resp. *k-coBüchi*) condition for $k \in \mathbb{N} \cup \{0\}$ if $\mathrm{occ}_\rho(q) \geq k$ for some $q \in \alpha$ (resp. $\mathrm{occ}_\rho(q) \leq k$ for every $q \in \alpha$).

The *language* of an automaton A, $\mathcal{L}(A)$, is the set of infinite words accepted by A. We say a *non-deterministic* automaton A accepts a word w when *some* run of A on w is accepting – that is, A has non-deterministic branching factor. In contrast, when A is a *universal* automaton *all* runs of A on w must be accepting – that is, A has universal branching factor.

Following the naming conventions described above, *Non-deterministic Büchi Word* (NBW) automata and *Non-deterministic k-Büchi Word* (NkBW) automata have non-deterministic branching factor and Büchi and k-Büchi acceptance condition, respectively. Similarly, *Universal Co-Büchi Word* (UCW) automata and *Universal k-coBüchi Word* (UkCW) automata have universal branching factor and coBüchi and k-coBüchi acceptance condition, respectively.

For clarity, throughout the paper we use a pairwise notation to denote the acceptance condition of automata. For example, $\mathcal{U}_k = (A, \mathrm{UkCW})$ and $\mathcal{N}_k = (A, NkBW)$ denote automaton A with, respectively, UkCW and NkBW acceptance conditions. This notation makes it easy to write automata that differ in their accepting conditions. In particular, it facilitates notation in complementation rules (cf. Proposition 1). The *complementation* of an automaton A is the task of constructing an automaton that accepts the words that are not in $\mathcal{L}(A)$.

Proposition 1. *The complementation of $\mathcal{U}_k$ is $\mathcal{N}_{k+1}$.*

2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) extends propositional logic with the unary temporal operator "next" and the binary temporal operator "until". As such, $\bigcirc\varphi$ stands for "next φ" and $\varphi \mathcal{U} \psi$ stands for "φ until ψ". LTL formulae over a set of propositions AP are evaluated over infinite sequences $s_0 s_1 \cdots$, where each s_i is a subset of AP. If $\sigma = s_0 s_1 \cdots$, then, for every $i \geq 0$:

- $\sigma, i \models \varphi$ if φ is a propositional formula and $s_i \models \varphi$.
- $\sigma, i \models \bigcirc\varphi$ iff $\sigma, i+1 \models \varphi$.
- $\sigma, i \models \varphi \mathcal{U} \psi$ iff there exists a $j \geq i$ such that $s_j \models \psi$ and for every $k \in \{i, \ldots, j-1\}$, $s_k \models \varphi$.

Finally we say that σ is a model of φ, denoted $\sigma \models \varphi$, iff $\sigma, 0 \models \varphi$. Three other common temporal operators, "eventually φ" ($\Diamond\varphi$), "always φ" ($\Box\varphi$), and "ψ releases φ" ($\psi \mathcal{R} \varphi$), can be defined as follows: $\Diamond\varphi \doteq \top \mathcal{U} \varphi$, $\Box\varphi \doteq \neg\Diamond\neg\varphi$, and $\psi \mathcal{R} \varphi \doteq \neg(\neg\psi \mathcal{U} \neg\varphi)$.

2.3 LTL and Automata

It is well-known that for an LTL formula φ one can construct an automaton $\mathcal{N}_\varphi = (A_\varphi, \mathrm{NBW})$ that accepts all and only the infinite words that satisfy φ. The construction is worst-case exponential in the size of the formula. An UCW automaton $\mathcal{U}_\varphi$ that accepts the models of φ can be obtained from the construction of an NBW automaton that accepts the models of $\neg\varphi$. Formally, $\mathcal{U}_\varphi = (A, \mathrm{UCW})$ accepts the models of φ iff $\mathcal{N}_{\neg\varphi} = (A, \mathrm{NBW})$ accepts the models of $\neg\varphi$ (cf. [Kupferman and Vardi, 2005]). Lemma 1 summarizes these results.

Lemma 1. *Given an LTL formula φ, it is possible to build an NBW automaton $\mathcal{N}_\varphi$, and a UCW automaton $\mathcal{U}_\varphi$, such that $\mathcal{L}(\mathcal{N}_\varphi) = \mathcal{L}(\mathcal{U}_\varphi) = \{w \in \Sigma^\omega \mid w \models \varphi\}$. The constructions are worst-case exponential in the size of φ.*

2.4 Fully Observable Non-Deterministic Planning

Here we briefly describe the Fully Observable Non-Deterministic (FOND) setting, and refer the interested reader to Geffner and Bonet [2013] for a more thorough description.

A *FOND problem* is a tuple $\langle F, O, I, G \rangle$, where F is a set of fluents (atomic propositions whose truth value can change over time); $I \subseteq F$ are the fluents true in the initial state; $G \subseteq F$, are the fluents that must hold in a goal state; and O is the set of (possibly non-deterministic) operators or action schemas. Each $o \in O$ is made up of Pre_o (the fluents that must hold for o to be executable) and Eff_o (the set of possible effects for o, one of which will update the state).

States are represented by a set of fluents (all those not in the set are presumed false) and partial states are represented by a set of literals (i.e., fluents or their negation). An action is a ground operator. The *outcome* of action a in state s is another state s' that results from updating s with one of the effects of the ground operator o corresponding to a. A distinguishing property of FOND planning is that the action outcome is not known until run time, and so all contingencies must be planned for. For this reason, solutions to a FOND problem take the form of a policy π that maps the state of the world to the action that should be executed. An execution of a policy π from state s_0 is a sequence of states $s_0, s_1, \ldots$ such that each s_{i+1} is an outcome of $\pi(s_i)$ in s_i.

Cimatti *et al.* [2003] defined different classes of solutions to a FOND problem. Here, we examine two of them. A *strong solution* is a policy that is guaranteed to achieve the goal regardless of non-determinism. *Strong cyclic solutions* are less restrictive and guarantee goal reachability in the presence of *fairness*, which presumes that all the action outcomes in a given state would manifest infinitely often. As the name suggests, a strong cyclic solution may revisit states. Strong cyclic solutions have the property stated in Proposition 2.

Proposition 2 (Strong Cyclic Solution)**.** *A policy π is a strong cyclic solution iff for every state s reachable by π there exists an execution that achieves the goal from s.*

Automated planning technology is highly optimized, and typically used for computation of plans or policies that achieve a prescribed goal after execution of a *finite number of actions* – that can be unbounded in strong cyclic plans. Patrizi *et al.* [2013] proposed a means of synthesizing infinite plans for planning domains with LTL goal formulae and *fair* non-deterministic actions. Their technique compiles the original domain into a new domain in a way that executions of a strong cyclic solution to the modified domain produce infinite plans in the original domain. The approach is limited to the class of LTL formulae that can be compiled into deterministic Büchi automata.

3 LTL Realizability and Synthesis

The *realizability* and *synthesis* problems for an LTL specification were first posed by Pnueli and Rosner in 1989. Since then, the scholarly literature has explored two slightly different interpretations of an LTL specification in the context of synthesis. In this paper, we study both interpretations, which differ in their semantics, distinguishing them via the subscript $s \in \{\text{Mealy}, \text{Moore}\}$ notation (cf. Definition 1). We adopt the terms *Mealy* and *Moore* semantics, which have been used previously in the synthesis literature (e.g. [Ehlers, 2011; Khalimov *et al.*, 2013]). A *strategy* for an LTL specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_s$ is a function $f : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ that maps *histories*, or finite sequences of subsets of $\mathcal{X}$, into subsets of $\mathcal{Y}$. A strategy f is *winning* when, for every infinite sequence $X_1 X_2 \cdots$ of subsets of $\mathcal{X}$, either: (i) s = Mealy and $\{(X_i \cup f(X_1 \cdots X_i))\}_{i \geq 1}$ satisfies φ; or (ii) s = Moore and $\{(X_i \cup f(X_0 \cdots X_{i-1}))\}_{i \geq 1}$ satisfies φ for some constant X_0, typically set to the empty trace ϵ. Winning strategies for an LTL specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Moore}}$ are also winning strategies for $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Mealy}}$, but the opposite does not hold in general (see example below). The *realizability* problem consists of determining existence of a winning strategy, and the *synthesis* problem consists of computing one (Definition 2). For unrealizable specifications, a *certificate of unrealizablity* is a strategy $g : (2^{\mathcal{Y}})^* \to 2^{\mathcal{X}}$ that prevents the agent from realizing φ. Lemma 2 formulates a well-known duality between the two semantics in LTL realizability.

Definition 1. *An* LTL specification *is a tuple $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_s$, where $\mathcal{X}$ and $\mathcal{Y}$ are two finite and disjoint sets of variables, φ is an* LTL *formula over variables in $\mathcal{X} \cup \mathcal{Y}$, and $s \in \{\textit{Mealy}, \textit{Moore}\}$.*

Definition 2. *The* realizability *problem for an* LTL *specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_s$ is to determine whether there exists a winning strategy. The* synthesis *problem for a realizable* LTL *specification consists of computing a winning strategy. We say that an* LTL *specification is* unrealizable *when it is not realizable.*

Lemma 2 (Duality)**.** *An* LTL *specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\textit{Mealy}}$ is realizable iff $\langle \mathcal{Y}, \mathcal{X}, \neg\varphi \rangle_{\textit{Moore}}$ is unrealizable.*

Example LTL specification $\langle \{x\}, \{y\}, \square(x \leftrightarrow y) \rangle_{\text{Mealy}}$ is realizable, and has a winning strategy f that maps each sequence $X_1 \cdots X_n \in (2^{\mathcal{X}})^*$ to $\{y\}$ if $X_n = \{x\}$, and to $\emptyset$ otherwise. On the other hand, $\langle \{x\}, \{y\}, \square(x \leftrightarrow y) \rangle_{\text{Moore}}$ is unrealizable. An unrealizability certificate $g : (2^{\mathcal{Y}})^* \to 2^{\mathcal{X}}$ can be constructed by assigning $g(\{y\}) = \emptyset$ and $g(\emptyset) = \{x\}$. Clearly, plays that start with a prefix $(g(f(X_0)) \cup f(X_0))$ cannot satisfy $\square(x \leftrightarrow y)$.

3.1 Automata Games

Following Thomas [1995], in this paper a *two-player game* is defined with respect to two finite sets of variables, $\mathcal{X}$ and $\mathcal{Y}$, and an automaton $\mathcal{A}$ with alphabet $\Sigma = 2^{\mathcal{X} \cup \mathcal{Y}}$ whose language describes which plays are winning in the game. In this paper we exploit the correspondence between LTL synthesis and games in various forms. To facilitate readability, in this paper we opt to represent games with a tuple $\langle \mathcal{X}, \mathcal{Y}, \mathcal{A} \rangle_s$, where $s \in \{\text{Moore}, \text{Mealy}\}$. In the following, we formalize automata games, and present duality results that are analogous to those for LTL synthesis.

We first describe the dynamics of a game $\langle \mathcal{X}, \mathcal{Y}, \mathcal{A} \rangle_{\text{Moore}}$, which corresponds to the definition of automata games commonly used in the literature. A game $\langle \mathcal{X}, \mathcal{Y}, \mathcal{A} \rangle_{\text{Moore}}$ has two players, *P1* and *P2*. In each round of the game, *P1* selects a subset of variables $Y_i \subseteq \mathcal{Y}$, followed by *P2*, which selects a subset of variables $X_i \subseteq \mathcal{X}$. The game is played an infinite number of rounds, and so a *play* consists of an infinite word $w = \{(X_i \cup Y_i) \mid X_i \subseteq \mathcal{X}, Y_i \subseteq \mathcal{Y}\}_{i \geq 1}$. The play is winning (for *P1*) if $\mathcal{A}$ accepts w, and the game is winning (for *P1*) if *P1* has a strategy that only yields winning plays, regardless of the moves of *P2*. Formally, $\langle \mathcal{X}, \mathcal{Y}, \mathcal{A} \rangle_{\text{Moore}}$ is *winning* if there exists an strategy $f : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ such that, for any sequence $\{X_i\}_{i \geq 1}$ of moves of *P2*, $\mathcal{A}$ accepts the infinite word $w = \{(X_i \cup f(X_0 \cdots X_{i-1}))\}_{i \geq 1}$ for some constant X_0.

In this paper we are also interested in examining the conditions under which player *P2* has a winning strategy. We write $\langle \mathcal{X}, \mathcal{Y}, \mathcal{A} \rangle_{\text{Mealy}}$ to denote a game with inverted turns with respect to $\langle \mathcal{X}, \mathcal{Y}, \mathcal{A} \rangle_{\text{Moore}}$. In each round, player *P1* selects a subset of variables of $\mathcal{X}$, followed by player *P2*, which selects a subset of variables of $\mathcal{Y}$. The difference now is

that the game is P2-centric. We say that the game is *winning* (for *P2*) if there exists an strategy $g : (2^X)^* \to 2^Y$ such that, for any sequence $\{X_i\}_{i\geq 1}$ of moves of *P1*, the play $w = \{(X_i \cup f(X_1 \cdots X_i))\}_{i\geq 1}$ is accepted by $\mathcal{A}$.

Proposition 3 below is analogous to Lemma 2, and formalizes the well-known *zero-sum* property of automata games. That is, player *P1* has a strategy to win a game $\langle X, Y, \mathcal{A}\rangle_{\text{Moore}}$ iff player *P2* has no strategy to win the *dual* game $\langle Y, X, \overline{\mathcal{A}}\rangle_{\text{Mealy}}$, where $\overline{\mathcal{A}}$ is an automaton that accepts the complement language of $\mathcal{A}$. Note that duality inverts the set of controllable variables. Throughout this paper we exploit a particular case of Proposition 3 with two classes of automata that recognize complement languages: UkCW and NkBW automata. We formalize the result in Lemma 3.

Proposition 3 (Duality)**.** *A game $\langle X, Y, \mathcal{A}\rangle_{Moore}$ is winning (for* P1*) iff the game $\langle Y, X, \overline{\mathcal{A}}\rangle_{Mealy}$ is not winning (for* P2*).*

Lemma 3 (Duality)**.** *Let $\mathcal{U}_k = (A, UkCW)$ and $\mathcal{N}_k = (A, NkBW)$. A game $\langle X, Y, \mathcal{U}_k\rangle_{Moore}$ is winning iff $\langle Y, X, \mathcal{N}_{k+1}\rangle_{Mealy}$ is not winning. Likewise, $\langle X, Y, \mathcal{U}_k\rangle_{Mealy}$ is winning iff $\langle Y, X, \mathcal{N}_{k+1}\rangle_{Moore}$ is not winning.*

3.2 LTL Realizability and Automata Games

The correspondence between LTL realizability and automata games has been commonly exploited to approach LTL realizability and synthesis (e.g. [Jobstmann and Bloem, 2006; Filiot *et al.*, 2009; Ehlers, 2010]). Theorem 1 formalizes the mapping, which preserves winning strategies. Typically, automata-based approaches to LTL synthesis rely on *deterministic* automata. NBW automata can be determinized (into Rabin automata) with the *Safra construction*. This construction, unfortunately, is very involved and not amenable to symbolic methods. Reportedly, the best tools cannot even handle relatively simple automata with more than 6 states.

Theorem 1. *Let φ be an* LTL *formula over $X \cup Y$, and let $\mathcal{A}_\varphi$ be an automaton that accepts the models of φ. σ is a winning strategy for $\langle X, Y, \varphi\rangle_{Moore}$ (resp. $\langle X, Y, \varphi\rangle_{Mealy}$) iff σ is a winning strategy for $\langle X, Y, \mathcal{A}_\varphi\rangle_{Moore}$ (resp. $\langle X, Y, \mathcal{A}_\varphi\rangle_{Mealy}$).*

Safety Games As an alternative to avoid Safra's construction, so-called *safraless* methods reduce the synthesis problem to a series of *safety* games over UkCW automata [Kupferman and Vardi, 2005]. In a safety game, all plays that violate the LTL specification formula φ have a *bad prefix*. A bad prefix is a finite trace π from which it is no longer possible to satisfy φ. That is, any infinite trace π' with prefix π is such that $\pi' \not\models \varphi$. The challenge in a safety game is for the agent to avoid bad prefixes. The advantage of using UkCW automata is that these can be determinized efficiently, using well-known methods derived from the *powerset construction* commonly used to determinize finite-state automata.

Reachability Games In a *reachability* game, winning plays have a *good prefix*, that is, a finite trace π such that any infinite trace π' with prefix π satisfies the specification. In particular, games over NkBW automata are reachability games.

4 Reductions to Automata Games

In this section we introduce a collection of techniques to reduce LTL realizability into a series of games defined over UkCW and NkBW automata. The reductions rely on Theorems 2, 3, 4, and 5, and are summarized in Table 1.

Theorem 2. *Let $\mathcal{A} = (A, UCW)$ be an automaton that accepts the models of $\neg\varphi$. The following are equivalent:*

- *$\langle X, Y, \varphi\rangle_{Mealy}$ is unrealizable*
- *$\langle Y, X, (A, UkCW)\rangle_{Moore}$ is winning for some k in $2^{O(|A|)}$*
- *$\langle X, Y, (A, NkBW)\rangle_{Mealy}$ not winning for some k in $2^{O(|A|)}$*

Proof sketch. By Lemma 2, $\langle X, Y, \varphi\rangle_{\text{Mealy}}$ is unrealizable iff $\langle Y, X, \neg\varphi\rangle_{\text{Moore}}$ is realizable. By well-known results on bounded synthesis (e.g. [Kupferman, 2006]), this occurs iff $\langle Y, X, (A, \text{UkCW})\rangle_{\text{Moore}}$ is winning for some k in $2^{O(|A|)}$. By Lemma 3, this occurs iff $\langle X, Y, (A, \text{NkBW})\rangle_{\text{Mealy}}$ is not winning for some k worst-case exponential in the size of A. □

Theorem 3. *Let $\mathcal{A} = (A, NBW)$ be an automaton that accepts the models of $\neg\varphi$. The following is equivalent:*

- *$\langle X, Y, \varphi\rangle_{Mealy}$ is realizable*
- *$\langle Y, X, (A, NkBW)\rangle_{Moore}$ is not winning for some k in $2^{O(|A|)}$*
- *$\langle X, Y, (A, UkCW)\rangle_{Mealy}$ is winning for some k in $2^{O(|A|)}$*

Proof sketch. By Lemma 2, $\langle X, Y, \varphi\rangle_{\text{Mealy}}$ is realizable iff $\langle Y, X, \neg\varphi\rangle_{\text{Moore}}$ is unrealizable. This occurs iff $\langle Y, X, (A, \text{NkBW})\rangle_{\text{Moore}}$ is not winning for some $k < \infty$. By Lemma 3, this occurs iff $\langle X, Y, (A, \text{UkCW})\rangle_{\text{Mealy}}$ is winning for some $k < \infty$. The results for bounded synthesis (e.g. [Kupferman, 2006]) set the exponential bound on k. □

Theorem 4. *Let $\mathcal{A} = (A, UCW)$ be an automaton that accepts the models of φ The following is equivalent:*

- *$\langle X, Y, \varphi\rangle_{Moore}$ is realizable*
- *$\langle X, Y, (A, UkCW)\rangle_{Moore}$ is winning for some k in $2^{O(|A|)}$*
- *$\langle Y, X, (A, NkBW)\rangle_{Mealy}$ is not winning for some k in $2^{O(|A|)}$*

Proof sketch. It follows from negating Theorem 2 and applying the duality results in Lemmas 2 and 3. □

Theorem 5. *Let $\mathcal{A} = (A, NBW)$ be an automaton that accepts the models of φ. The following is equivalent:*

- *$\langle X, Y, \varphi\rangle_{Moore}$ is unrealizable*
- *$\langle X, Y, (A, NkBW)\rangle_{Moore}$ is not winning for some k in $2^{O(|A|)}$*
- *$\langle Y, X, (A, UkCW)\rangle_{Mealy}$ is winning for some k in $2^{O(|A|)}$*

Proof sketch. It follows frin negating Theorem 3 and applying the duality results in Lemmas 2 and 3. □

Bounded Realizability and Synthesis Table 1 summarizes the results of the theorems above into a series of tests to determine realizability and unrealizability of an LTL specification. The tests based on reductions to UkCW games (i.e. safety games) are well known, and commonly referred to as *bounded synthesis* methods [Schewe and Finkbeiner, 2007], which build upon *safraless methods* [Kupferman and Vardi, 2005]. These tests are *constructive*, meaning that a winning strategy for the reduced game is a winning strategy for the original specification – in case it is realizable –, or a certificate of unrealizability – in case it is unrealizable. Bounded synthesis has been a successful approach to synthesis, and nowadays many modern tools exploit reductions to UkCW games in some way (e.g. *Lily* [Jobstmann and Bloem, 2006], *Unbeast* [Ehlers, 2010], and *Acacia/Acacia*$^+$ [Filiot *et al.*, 2009;

Specification	Construction of A	Test	Good for	Constructive	Thm
$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Mealy}}$	$\mathcal{L}(A, \text{NBW}) = \mathcal{L}(\varphi)$	$\exists k. \langle \mathcal{Y}, \mathcal{X}, (A, \text{UkCW}) \rangle_{\text{Moore}}$ winning for *P1* $\iff$ unrealizable	co-safe	yes	2
$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Mealy}}$	$\mathcal{L}(A, \text{NBW}) = \mathcal{L}(\varphi)$	$\exists k. \langle \mathcal{X}, \mathcal{Y}, (A, \text{NkBW}) \rangle_{\text{Mealy}}$ not winning for *P2* $\iff$ unrealizable	co-safe	no	2
$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Mealy}}$	$\mathcal{L}(A, \text{UCW}) = \mathcal{L}(\varphi)$	$\exists k. \langle \mathcal{Y}, \mathcal{X}, (A, \text{NkBW}) \rangle_{\text{Moore}}$ not winning for *P1* $\iff$ realizable	safe	no	3
$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Mealy}}$	$\mathcal{L}(A, \text{UCW}) = \mathcal{L}(\varphi)$	$\exists k. \langle \mathcal{X}, \mathcal{Y}, (A, \text{UkCW}) \rangle_{\text{Mealy}}$ winning for *P2* $\iff$ realizable	safe	yes	3
$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Moore}}$	$\mathcal{L}(A, \text{UCW}) = \mathcal{L}(\varphi)$	$\exists k. \langle \mathcal{X}, \mathcal{Y}, (A, \text{UkCW}) \rangle_{\text{Moore}}$ winning for *P1* $\iff$ realizable	safe	yes	4
$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Moore}}$	$\mathcal{L}(A, \text{UCW}) = \mathcal{L}(\varphi)$	$\exists k. \langle \mathcal{Y}, \mathcal{X}, (A, \text{NkBW}) \rangle_{\text{Mealy}}$ not winning for *P2* $\iff$ realizable	safe	no	4
$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Moore}}$	$\mathcal{L}(A, \text{NBW}) = \mathcal{L}(\varphi)$	$\exists k. \langle \mathcal{X}, \mathcal{Y}, (A, \text{NkBW}) \rangle_{\text{Moore}}$ not winning for *P1* $\iff$ unrealizable	co-safe	no	5
$\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Moore}}$	$\mathcal{L}(A, \text{NBW}) = \mathcal{L}(\varphi)$	$\exists k. \langle \mathcal{Y}, \mathcal{X}, (A, \text{UkCW}) \rangle_{\text{Mealy}}$ winning for *P2* $\iff$ unrealizable	co-safe	yes	5

Table 1: Strategies to determine realizability and unrealizability of an LTL specification via reduction to automata games. When φ is a safe (resp. co-safe) LTL formula, strategies that are *good for safe* (resp. *good for co-safe*) formulae only require tests with $k = 0$ in UkCW games; and $k = 1$ in NkBW games. *Constructive* tests yield a winning strategy or certificate, as appropriate.

Bohy *et al.*, 2012]). The tests based on reductions to NkBW games (i.e. reachability games) are novel. These tests determine realizability and unrealizability. By way of analogy to bounded synthesis, we refer to them as *bounded realizability* methods. Note that a complete approach for LTL realizability and synthesis would combine at least two of these approaches in parallel or with interleaved search.

Safe and Co-Safe LTL Specifications Different syntactic characterizations of LTL formulas have been studied (e.g. [Sistla, 1994]). Here, we examine the *safe* and *co-safe* fragment of LTL [Kupferman and Vardi, 2001]. Safe properties assert that something bad never happens. Co-safe properties are dual, and assert that something good eventually happens. The following syntactic fragments of LTL express safe, and co-safe properties:

$$\text{Safe: } \varphi = \top \mid \bot \mid p \mid \neg p \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{R} \varphi$$

$$\text{Co-safe: } \varphi = \top \mid \bot \mid p \mid \neg p \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi$$

Safe and co-safe LTL formulae can be transformed into automata with absorbing rejecting and accepting states, respectively [Kupferman and Vardi, 2001]. We observe that with these automata constructions, realizability of safe and co-safe formulae can be determined by performing only one test (Theorem 6).

Theorem 6. LTL *realizability for safe (resp. co-safe) specifications can be determined with strategies that are* good for safe *(resp.* good for co-safe*) formulae (cf. Table 1) by performing a single test with $k = 0$, in the case of reductions to UkCW games, and $k = 1$, in the case of reductions to NkBW games.*

Proof sketch. The proof follows from applying Lemma 3 in Theorems 2, 3, 4, and 5, with a construction of an NBW with absorbing accepting states. Here, we sketch the proof of one case. Let φ be a co-safe LTL formula, and let $\mathcal{A} = (A, \text{NBW})$ be an automaton that accepts the models of φ. $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_{\text{Moore}}$ is realizable iff $\langle \mathcal{X}, \mathcal{Y}, (A, \text{NkBW}) \rangle_{\text{Moore}}$ is winning for all k, but this happens iff $\langle \mathcal{X}, \mathcal{Y}, (A, \text{NkBW}) \rangle_{Moore}$ is winning for $k = 1$ (because A has absorbing accepting states). Lemma 3 derives that this only happens iff $\langle \mathcal{Y}, \mathcal{X}, (A, \text{UkCW}) \rangle_{\text{Mealy}}$ is not winning for $k = 0$. □

5 Automata Games via Planning

In recent years there has been increased interest in the development of algorithms that exploit automated planning techniques to synthesize programs of varying sorts including so-called *generalized planning* (e.g., [Aguas *et al.*, 2016; Bonet *et al.*, 2017]) and LTL synthesis. Camacho *et al.* [2018b] studied the correspondence between LTL synthesis and planning, and presented an algorithm to synthesize strategies, via planning, that is complete when the formula is transformed into deterministic Büchi automata, but not in general. Together with this work, it constitutes the first approach to LTL synthesis via planning. Also related to this work is [Camacho *et al.*, 2018a], where planning technology is used to solve LTL synthesis for LTL interpreted over *finite* words [De Giacomo and Vardi, 2015]. Approaching LTL synthesis as planning enables the use of highly optimized search techniques and heuristics. Planning algorithms can reason about action costs, stochasticity, plan quality, and preferences. Our work opens the door to using planning technology for richer notions of LTL synthesis, including those that optimize for *high-quality* strategies [Almagor and Kupferman, 2016].

In this section we present reductions of UkCW and NkBW games to FOND planning. These reductions comprise the first complete approach to LTL realizability and synthesis via planning. For an LTL specification $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, and a test strategy from Table 1, our approach consists of three steps:

(1) construct an automaton $\mathcal{A}$
(2) solve a sequence of automata games via FOND planning
(3) determine realizability or unrealizability

In addition, and if the test is constructive, a fourth step can be performed to synthesize a winning strategy – if realizable – or a certificate of unrealizability – if unrealizable.

Step 1: LTL to automata In the first step we construct an automaton (A, NBW) that accepts the models of φ or $\neg\varphi$, according to the strategy selected from Table 1. Our FOND compilation requires post-processing of the set of automaton transitions. Each transition $t = (q, \text{guard}(t), q')$ in A with *guard* $\text{guard}(t) = \bigvee_i c_i$ in DNF is decomposed into a set of transitions $t_i = (q, c_i, q')$. Each t_i has guard c_i, that is a conjunctive formula over variables in $\mathcal{X} \cup \mathcal{Y}$. For each t_i, we denote by $\text{macro}(t_i)$ the set of transitions derived from t. It can be proved that the decomposition of the transitions does not affect the language accepted by the automaton.

Step 2: Construction of a FOND Problem The second step takes as input the automaton generated in Step 1, and constructs a series of FOND problems $\mathcal{P}_k$, for $k \geq 0$. Each $\mathcal{P}_k$

simulates a game over a UkCW or NkBW automaton in accordance with the strategy selected from Table 1. We detail the FOND compilations for UkCW and NkBW automata games in Sections 5.1 and 5.2.

Step 3: Determine Realizability or Unrealizability The compiled FOND problems $\mathcal{P}_k$ have the following property: $\mathcal{P}_k$ has a solution iff the simulated automata game is winning. This property is exploited in the following manner: our technique searches for existence of solutions to the FOND problems $\mathcal{P}_k$ for $k = 0, 1, \ldots$ until the test in Table 1 is passed. Note that the tests are not guaranteed to terminate if, for instance, LTL realizability for an unrealizable specification is checked. However, termination can be achieved by combining two strategies in parallel: one that checks realizability, and another that checks unrealizability.

Step 4: Construction of a strategy A finite state controller that implements a strategy that solves the synthesis p roblem for specification φ can be constructed from a policy for $\mathcal{P}_k$. See [Patrizi *et al.*, 2013; Camacho *et al.*, 2017] for details.

5.1 UkCW Games via Planning

Figure 1 describes the details of the compilation of a game with automaton $A = \langle Q, \Sigma, q_0, \delta, \alpha \rangle$ into a FOND planning problem $\mathcal{P}_k$. For illustration purposes, we assume the game is in the form $\langle \mathcal{X}, \mathcal{Y}, (A, \text{UkCW}) \rangle_{\text{Mealy}}$, that is, we are searching for winning strategies for *P2*. At the end of the section we show how the dynamics of $\mathcal{P}_k$ can be modified to compile games in the form $\langle \mathcal{X}, \mathcal{Y}, (A, \text{UkCW}) \rangle_{\text{Moore}}$.

The dynamics of $\mathcal{P}_k$ simulates an infinite sequence of rounds in the two-player game. Each round is divided into three sequential stages. The first stage simulates the move of the environment and agent players. The move of *P1* is simulated with a cascade of actions that *non-deterministically* assign a value to the variables in $\mathcal{X}$. In turn, the moves of *P2* are simulated with a cascade of *deterministic* actions that assign a value to the variables in $\mathcal{Y}$. As a consequence of the player moves, some automaton transitions are deemed unfeasible. The second stage performs all automaton transitions. Fluents $\texttt{F(q)}$ are true in a planning state if there exists a run of A on the play being simulated that ends in q. The third stage sets the value of the counters associated to each automaton state q. A fluent $\texttt{newCnt(q,m)}$ is true if there exists a run of A on the simulated play that ends in q and hits m accepting states. The fluent $\texttt{oldCnt(q,m)}$ keeps track of the maximum number of hits for a fluent q. At the end of the third stage, the first stage is reestablished to simulate the next round.

There are two important things to notice about the dynamics of $\mathcal{P}_k$. First, when an input word being simulated is not accepted by (A, UkCW) (that is, a run visits more than k accepting states), a deadend in $\mathcal{P}_k$ is incurred. This is because a fluent $\texttt{newCnt(q,k+1)}$ prevents $\texttt{syncF}$ actions to disable the fluent $\texttt{F(q)}$, action $\texttt{continue}$ is no longer applicable, and the first stage can no longer be reestablished. Second, the non-determinism of action $\texttt{continue}$ ensures that execution of strong-cyclic solutions yield an infinite number of rounds in the game being simulated (cf. [Patrizi *et al.*, 2013]).

Finally, the turn-taking of player moves mandated by the semantics of the specification (Moore vs. Mealy) is forced by establishing a fixed order in the variables $v \in \mathcal{X} \cup \mathcal{Y}$. E.g. with the Moore semantics, the order must guarantee that the fluents $\texttt{turn(v}_\texttt{i}\texttt{)}$ first iterate over all variables in $\mathcal{Y}$.

Components of the SAFE2FOND compilation:

$$F \coloneqq \{\texttt{turn(v}_\texttt{i}\texttt{)}\}_{0 \le i \le |\mathcal{X}\cup\mathcal{Y}|} \cup \{\texttt{sync}, \texttt{goal}\} \cup \{\texttt{poss(t)}\}_{t \in T} \cup \{\texttt{oldCnt(q,m)}, \texttt{newCnt(q,m)}\}_{q \in Q, 0 \le m \le k}$$

$$I \coloneqq \{\texttt{sync}, \texttt{oldCnt}(\texttt{q}_\texttt{0}, \texttt{0})\} \cup \{\texttt{safe(t)}\}_{t \in \text{Safe}(T)} \cup \{\texttt{poss(t)}\}_{t.orig = q_0}$$

$$\text{Safe}(T) \coloneqq \{(q, \sigma, q') \in T \mid q' \notin \alpha\}$$

$$O \coloneqq \{\texttt{moveX(v)}\}_{v \in \mathcal{X}} \cup \{\texttt{movePosY(v)}, \texttt{moveNegY(v)}\}_{v \in \mathcal{Y}} \cup \{\texttt{transAcc(t,m)}, \texttt{transRej(t,m)}\}_{t \in T, 0 \le m \le k} \cup \{\texttt{startSync}, \texttt{continue}\} \cup \{\texttt{syncF(q,m)}\}_{q \in Q, 0 \le m \le k}$$

$$G \coloneqq \{\texttt{goal}\}$$

Stage 1: Actions that simulate $\mathcal{X}$ and $\mathcal{Y}$ moves:

$$Pre_{\texttt{moveX(v}_\texttt{i}\texttt{)}} = \{\texttt{turn(v}_\texttt{i}\texttt{)}\}$$

$$Eff_{\texttt{moveX(v}_\texttt{i}\texttt{)}} = \{\texttt{turn(v}_{\texttt{i+1}}\texttt{)}, \neg\texttt{turn(v}_\texttt{i}\texttt{)}\} \cup oneof(e_1, e_2)$$

$$Pre_{\texttt{movePosY(v}_\texttt{i}\texttt{)}} = \{\texttt{turn(v}_\texttt{i}\texttt{)}\}$$

$$Eff_{\texttt{movePosY(v}_\texttt{i}\texttt{)}} = \{\texttt{turn(v}_{\texttt{i+1}}\texttt{)}, \neg\texttt{turn(v}_\texttt{i}\texttt{)}\} \cup e_1$$

$$Pre_{\texttt{moveNegY(v}_\texttt{i}\texttt{)}} = \{\texttt{turn(v}_\texttt{i}\texttt{)}\}$$

$$Eff_{\texttt{moveNegY(v}_\texttt{i}\texttt{)}} = \{\texttt{turn(v}_{\texttt{i+1}}\texttt{)}, \neg\texttt{turn(v}_\texttt{i}\texttt{)}\} \cup e_2$$

$$e_1 = \{\neg\texttt{poss(t)}\}_{t \in T, \neg v_i \in \text{Lits(guard}(t))}$$

$$e_2 = \{\neg\texttt{poss(t)}\}_{t \in T, v_i \in \text{Lits(guard}(t))}$$

Stage2: Actions that simulate transitions, $t \in T$, $0 \le m \le k$:

$$Pre_{\texttt{transRej(t,m)}} = \{\texttt{turn(v}_{|\mathcal{X}\cup\mathcal{Y}|}\texttt{)}, \texttt{poss(t)}, \texttt{safe(t)}, \texttt{oldCnt(q,m)}\}$$

$$Eff_{\texttt{transRej(t,m)}} = \{\texttt{F(q')}, \texttt{newCnt(q',m)}\} \cup \{\neg\texttt{poss(t')}\}_{t' \in \text{macro}(t)}$$

$$Pre_{\texttt{transAcc(t,m)}} = \{\texttt{turn(v}_{|\mathcal{X}\cup\mathcal{Y}|}\texttt{)}, \texttt{poss(t)}, \neg\texttt{safe(t)}, \texttt{oldCnt(q,m)}\}$$

$$Eff_{\texttt{transAcc(t,m)}} = \{\texttt{F(q')}, \texttt{newCnt(q',m+1)}\} \cup \{\neg\texttt{poss(t')}\}_{t' \in \text{macro}(t)}$$

Stage 3: Actions that synchronize the automaton, $q \in Q$, $0 \le m \le k$:

$$Pre_{\texttt{startSync}} = \{\texttt{turn(v}_{|\mathcal{X}\cup\mathcal{Y}|}\texttt{)}\} \cup \{\neg\texttt{poss(t)}\}_{t \in T}$$

$$Eff_{\texttt{startSync}} = \{\texttt{sync}, \neg\texttt{turn(v}_{|\mathcal{X}\cup\mathcal{Y}|}\texttt{)}\}\,\{\neg\texttt{oldCnt(q,m)}\}_{q \in Q, 0 \le m \le k}$$

$$Pre_{\texttt{syncF(q,m)}} = \{\texttt{sync}, \texttt{F(q)}, \texttt{newCnt(q,m)}\} \cup \{\neg\texttt{newCnt(q,n)}\}_{m < n \le k}$$

$$Eff_{\texttt{syncF(q,m)}} = \{\texttt{poss(t)}\}_{t.orig = q} \cup \{\texttt{oldCnt(q,m)}\} \cup \{\neg\texttt{F(q)}\} \cup \{\neg\texttt{newCnt(q,n)}_{0 \le n \le k}\}$$

Action that reestablishes the dynamics of the problem:

$$Pre_{\texttt{continue}} = \{\texttt{sync}\} \cup \{\neg\texttt{F(q)}\}_{q \in Q}$$

$$Eff_{\texttt{continue}} = oneof(e_3, e_4)$$

$$e_3 = \{\texttt{goal}\}$$

$$e_4 = \{\texttt{turn(v}_\texttt{0}\texttt{)}, \neg\texttt{sync}\}$$

Figure 1: Details of the SAFE2FOND compilation for a game $\mathcal{S} = \langle \mathcal{X}, \mathcal{Y}, (A, \text{UkCW}) \rangle_s$ with automaton $A = \langle Q, \Sigma, q_0, \delta, \alpha \rangle$.

SAFE2FOND($\mathcal{S}$) denotes the FOND problem constructed from specification $\mathcal{S} = \langle \mathcal{X}, \mathcal{Y}, (A, \text{UkCW}) \rangle_s$, where $s \in \{\text{Moore}, \text{Mealy}\}$. The construction of SAFE2FOND($\mathcal{S}$) is polynomial, and thus existence of a winning strategy for $\mathcal{S}$ can be determined in time that is exponential in the size of A. Recall that FOND planning is EXPTIME-complete in the size of succinct representations of the problem [Rintanen, 2004].

Theorem 7. *The construction* SAFE2FOND($\mathcal{S}$) *is a polynomial reduction of UkCW games $\mathcal{S}$ into FOND planning. If π is a strong cyclic solution to* SAFE2FOND($\mathcal{S}$) *then a winning strategy for $\mathcal{S}$ can be constructed from π.*

Proof sketch. The proof of correctness for UkCW-based

automata compilations follows the intuition explained below. The dynamics of the automaton are captured by planning actions. Each planning state s simulates an automaton macrostate (as in the powerset construction), and has the property that $\texttt{F(q)} \in s$ iff there exists a run on the automaton in the simulated play that ends in q. In addition, planning states count the maximum number of times automaton runs hit accepting states. Given the properties above, and a fixed k, the UkCW game has a winning strategy iff there exists a policy that (1) only visits planning states with associated counter that is not greater than k, and (2) runs in perpetuity. Property (1) is achieved by forcing planning states with associated counter equal to k to be deadends – and this is actually done in the current compilation. Property (2) is achieved by strong-cyclic solutions, with the trick of introducing the artificial non-deterministic action `continue` and dummy goals (cf. [Patrizi *et al.*, 2013]). The construction of SAFE2FOND($\mathcal{S}$) is polynomial. A transducer that implements a winning strategy can be constructed by unfolding a strong cyclic solution, collapsing states that belong to the same simulated turn. □

5.2 NkBW Games via Planning

The construction SAFE2FOND presented in Section 5.1 can be modified into an algorithm, REACH2FOND, that solves NkBW games via reduction to strong FOND planning. The main modifications follow.

In contrast to safety games, solutions to a reachability game have to yield good prefixes. Recall that the compiled FOND problem has fluents $\texttt{newCnt(q,m)}$ that keep track of whether there exists a run of A on the play being simulated that hits m accepting states. In a NkBW reachability game, a good prefix is achieved when a state in which $\texttt{newCnt(q,k)}$ holds is visited. As such, in REACH2FOND($\mathcal{S}$) the goal is no longer the dummy fluent `goal`, but to reach one of the fluents $\texttt{newCnt(q,k)}$ for $q \in Q$. The action `continue` is now deterministic, and its unique effect reestablishes the first stage. This time, not all runs of the automaton need to be carried over. To this end, action `continue` does not require all $\texttt{F(q)}$ to be false, and instead falsifies all of them. Finally, *strong* solutions to REACH2FOND($\mathcal{S}$) yield winning strategies for $\mathcal{S}$. The correctness and complexity results obtained in Section 5.1 extend to FOND compilations of NkBW games.

Theorem 8. *The construction* REACH2FOND($\mathcal{S}$) *is a polynomial reduction of NkBW games $\mathcal{S}$ into FOND planning. If π is a strong solution to* REACH2FOND($\mathcal{S}$) *then a winning strategy for $\mathcal{S}$ can be constructed from π.*

Proof sketch. Similar to the proof of correctness of Theorem 7, the dynamics of REACH2FOND($\mathcal{S}$) simulate plays of the game. This time, the dynamics simulate transitions in a NBW automaton. Planning states can do bookkeeping of *some* runs of the NBW (not necessarily all), and count the maximum number of times that these runs hit accepting states. In other words, a fixed play can be simulated by carrying over *some* runs of the NBW, or *all*, at the discretion of the planner. As such, strong solutions to REACH2FOND($\mathcal{S}$) yield winning strategies to $\mathcal{S}$. A transducer that implements a winning strategy can be constructed by unfolding a strong solution, collapsing states that belong to the same simulated turn. In the other direction, winning strategies can be simulated with the dynamics of REACH2FOND($\mathcal{S}$). The construction of REACH2FOND($\mathcal{S}$) is polynomial. □

6 Multiple Automata Decompositions

Our automata-based approaches to LTL synthesis rely on worst-case exponential transformations of the LTL formula into NBW and UCW automata. In practice, this complexity can be reduced by systematically decomposing LTL formula φ into smaller subformulae and transforming them into multiple automata that together capture the models of φ. Such transformations have been exploited in some previous approaches to LTL synthesis (e.g., [Camacho *et al.*, 2017; Bohy *et al.*, 2012; Camacho *et al.*, 2018a]) and in approaches to planning with LTL goals and preferences (e.g. [Baier and McIlraith, 2006]).

In Acacia$^+$, specification φ is systematically decomposed into the conjunction of a set of subformulae φ_i, each of which is transformed into a UCW $\mathcal{A}_i$. The resulting game is the composition of potentially smaller safety subgames, each one with associated UkCW $\mathcal{A}_i$, which run in parallel. Winning strategies have to avoid bad prefixes in *all* of the subgames.

In [Camacho *et al.*, 2018a], we presented a reduction of LTL_f synthesis – that is, synthesis of LTL specifications interpreted over *finite* words – into FOND planning. The construction of a FOND problem is similar to the NkBW-based compilation presented in this paper, with the major exception that the target was non-deterministic finite-state automata (NFA). Automata decompositions take a specification φ in NNF, decompose it into a conjunction of subformulae φ_i, and transform each φ_i into an NFA $\mathcal{A}_i$. The resulting game is the composition of potentially smaller subgames, each one with associated NFA $\mathcal{A}_i$, that run in parallel. The dynamics of all subgames are integrated within a single FOND problem, and the goal is achieved whenever *all* subgames are winning.

The FOND compilations that we presented in Section 5 are also amenable to automata decompositions. FOND compilations of UkCW games can use multiple automata decompositions following Acacia$^+$, whereas NkBW games can use automata decompositions following Camacho *et al.* [2018a].

7 Evaluation

We implemented the algorithms for LTL realizability and synthesis via FOND in a tool we named *SynKit* [Camacho *et al.*, 2018c]. *SynKit* implements the algorithms presented in this paper, and others (e.g., those in [Camacho *et al.*, 2018a] for finite LTL synthesis). We use *Spot* to transform LTL formulae into NBW [Duret-Lutz *et al.*, 2016]. Reductions to automata games listed in Table 1 are solved via FOND planning with SAFE2FOND and REACH2FOND compilations. We use PRP to generate strong cyclic solutions [Muise *et al.*, 2012], and MyND to generate strong solutions [Mattmüller *et al.*, 2010]. Our experiments were run on an Intel Xeon E5-2430 2.2GHz processor, and each process was limited to 30 minute run times and 4GB memory usage. We configured *SynKit* to perform reductions to UkCW games with $k \leq 2$ and NkBW games with $k \leq 3$.

	Real.		Unreal.							
Benchmark	NkBW	UkCW	NkBW	UkCW	SynKit	Acacia$^+$	Bowser	Bosy	ltlsynt	Party
Amba Decomposed (23)	15	14	–	–	15	14	18	22	22	22
Detector (6)	1	2	–	–	2	4	3	4	3	6
Detector Unreal (6)	–	–	4	5	5	0	4	6	3	6
Full Arbiter (6)	1	2	–	–	2	4	2	2	5	6
Full Arbiter Unreal (12)	–	–	9	9	9	0	8	12	12	12
Genbuf (5)	0	0	–	–	0	4	0	0	0	3
Generalized Buffer (5)	0	0	–	–	0	5	0	0	0	3
Lilydemo (24)	16	16	5	5	21	19	24	24	23	24
Load Balancer (5)	1	2	–	–	2	2	2	2	3	4
Load Balancer Unreal (12)	–	–	11	11	11	0	7	11	11	11
Loadcomp (4)	1	2	–	–	2	4	4	4	4	4
Loadfull (4)	1	2	–	–	2	3	4	4	4	4
LTL2DBA (27)	18	23	–	–	23	26	24	24	27	27
LTL2DPA (24)	17	23	–	–	23	23	24	24	23	24
Prio Arbitrer (6)	1	1	–	–	1	5	3	3	4	5
Prio Arbitrer Unreal (4)	–	–	1	1	1	0	3	3	3	3
RR Arbiter (6)	1	1	–	–	1	3	3	3	5	2
RR Arbiter Unreal (4)	–	–	1	2	2	0	3	2	3	3
Simple Arbiter (6)	1	2	–	–	2	4	3	4	4	5
Simple Arbiter Unreal (11)	–	–	2	2	2	0	6	8	9	8

Table 2: Number of problems solved by *SynKit* on a collection of benchmarks used in SYNTCOMP 2017. Realizability (Real.) and unrealizability (Unreal.) tests were performed via reductions to NkBW (i.e., safety) and UkCW (i.e., reachability) games. Safety and reachability games were solved via FOND planning with Safe2FOND and Reach2FOND compilations, respectively. For reference and broad comparison, results for tools that participated at SYNTCOMP 2017 are also reported. Note that these results were run under more favorable computational settings.

Synthesis via Planning One of the main objectives of our experiments was to evaluate whether automated planning is a viable technology to address LTL synthesis. We tested the performance of our planning-based methods over a collection of benchmarks used in the sequential realizability track of the SYNTCOMP 2017 competition. Table 2 summarizes the aggregated coverage results (i.e., number of problems solved) obtained by *SynKit*. For reference, we include coverage results of participants in SYNTCOMP 2017: *Party*, *ltlsynt*, *Bosy*, *Bowser*, and *Acacia*$^+$. These results are reported on the competition website. They were run with an Intel XEON E3-1271 3.6GHz processor, 32GB of memory, and a processing time limit of 60 minutes. Rather than giving a precise comparison of run time, our objective was to evaluate whether planning technology, and the algorithms presented in this paper are competitive with more mature synthesis tools. In terms of run time, we observed that most problems can be solved with low time and memory usage. In terms of coverage, while *SynKit* does not outperform any of the tools we compared with, it is very competitive and able to determine realizability for several problems that eluded other solvers.

Advantage of NkBW-based Reductions The second objective of our experiments was to assess the advantage of our novel *bounded realizability* methods over existing *bounded synthesis*. Table 2 details the coverage results obtained with the tests for determining realizability and unrealizability listed in Table 1. The coverage obtained with reductions to NkBW and UkCW games is very similar. We observed that the run times differ significantly in some problems, and no method clearly dominates the other. Thus, bounded realizability can be an alternative to bounded synthesis.

8 Summary and Discussion

LTL synthesis is central to the automated construction of controllers and certain classes of programs. While LTL synthesis is a well-established problem, it has only been in the last decade that efficient tools have started to emerge. Common approaches to LTL synthesis reduce the problem to automata games and many modern tools make use of bounded synthesis techniques based on safety games played over UkCW.

In this paper we established the duality between so-called Mealy and Moore semantics of LTL specifications in the context of LTL synthesis. We exploited this duality to define novel reductions of LTL realizability to reachability games played over NkBW automata. We also introduced the first complete approach to LTL realizability and synthesis via automated planning. Our approach compiles automata games into instances of FOND planning. Preliminary experimental results show that reductions to NkBW games can be beneficial and complement other well-known reductions to UkCW games. Moreover, they show that planning technology can be a competitive technique to approach synthesis.

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

[Aguas *et al.*, 2016] Javier Segovia Aguas, Sergio Jiménez Celorrio, and Anders Jonsson. Generalized planning with procedural domain control knowledge. In *ICAPS*, pages 285–293, 2016.

[Almagor and Kupferman, 2016] Shaull Almagor and Orna Kupferman. High-quality synthesis against stochastic environments. In *CSL*, pages 28:1–28:17, 2016.

[Baier and McIlraith, 2006] Jorge A. Baier and Sheila A. McIlraith. Planning with temporally extended goals using heuristic search. In *ICAPS*, pages 342–345, 2006.

[Bohy *et al.*, 2012] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In *CAV*, 2012.

[Bonet *et al.*, 2017] Blai Bonet, Giuseppe De Giacomo, Hector Geffner, and Sasha Rubin. Generalized planning: Non-deterministic abstractions and trajectory constraints. In *IJCAI*, pages 873–879, 2017.

[Camacho *et al.*, 2017] Alberto Camacho, Eleni Triantafillou, Christian J. Muise, Jorge A. Baier, and Sheila A. McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*, pages 3716–3724, 2017.

[Camacho *et al.*, 2018a] Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith. Finite LTL synthesis as planning. In *ICAPS*, 2018. To appear.

[Camacho *et al.*, 2018b] Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith. Synthesizing controllers: On the correspondence between LTL synthesis and non-deterministic planning. In *Advances in Artificial Intelligence – Proceedings of the 31st Canadian Conference on Artificial Intelligence*, pages 45–59, 2018.

[Camacho *et al.*, 2018c] Alberto Camacho, Christian J. Muise, Jorge A. Baier, and Sheila A. McIlraith. SynKit: LTL synthesis as a service. In *IJCAI*, 2018. To appear.

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

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, pages 1558–1564, 2015.

[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 ω-automata manipulation. In *ATVA*, pages 122–129, 2016.

[Ehlers, 2010] Rüdiger Ehlers. Symbolic bounded synthesis. In *CAV*, volume 6174, pages 365–379. Springer, 2010.

[Ehlers, 2011] Rüdiger Ehlers. Experimental aspects of synthesis. In *IWIGP*, pages 1–16, 2011.

[Filiot *et al.*, 2009] Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. An antichain algorithm for LTL realizability. In *CAV*, pages 263–277, 2009.

[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.

[Jacobs *et al.*, 2016] Swen Jacobs, Felix Klein, and Sebastian Schirmer. A high-level LTL synthesis format: TLSF v1.1. In *SYNT*, pages 112–132, 2016.

[Jacobs *et al.*, 2017] Swen Jacobs, Nicolas Basset, Roderick Bloem, Romain Brenguier, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Thibaud Michaud, Guillermo A. Pérez, Jean-François Raskin, Ocan Sankur, and Leander Tentrup. The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants & results. In *SYNT*, pages 116–143, 2017.

[Jobstmann and Bloem, 2006] Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *FMCAD*, pages 117–124, 2006.

[Khalimov *et al.*, 2013] Ayrat Khalimov, Swen Jacobs, and Roderick Bloem. PARTY parameterized synthesis of token rings. In *CAV*, pages 928–933. Springer, 2013.

[Kupferman and Vardi, 2001] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[Kupferman and Vardi, 2005] Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *FOCS*, pages 531–542, 2005.

[Kupferman, 2006] Orna Kupferman. Avoiding determinization. In *LICS*, pages 243–254, 2006.

[Mattmüller *et al.*, 2010] Robert Mattmüller, Manuela Ortlieb, Malte Helmert, and Pascal Bercher. Pattern database heuristics for fully observable nondeterministic planning. In *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 *ICAPS*, 2012.

[Patrizi *et al.*, 2013] Fabio Patrizi, Nir Lipovetzky, and Hector Geffner. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *IJCAI*, 2013.

[Pnueli and Rosner, 1989] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of POPL*, pages 179–190, 1989.

[Rintanen, 2004] Jussi Rintanen. Complexity of planning with partial observability. In *ICAPS*, pages 345–354, 2004.

[Schewe and Finkbeiner, 2007] Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *ATVA*, pages 474–488, 2007.

[Sistla, 1994] A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.

[Thomas, 1995] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *STACS*, pages 1–13. Springer, 1995.