Automaton-Guided Controller Synthesis for Nonlinear Systems with Temporal Logic

Eric M. Wolff, Ufuk Topcu, and Richard M. Murray

Abstract—We develop a method to synthesize control policies for discrete-time nonlinear dynamical systems subject to temporal logic specifications. Our approach uses an approximate abstraction of the system along with a automaton representing the temporal logic specification to guide the search for a feasible control policy. This approach decomposes the search for a feasible control policy into a series of constrained reachability problems, and can thus be applied to any system for which (possibly approximate) solutions to constrained reachability problems are computable. Examples of methods for solving constrained reachability problems include sampling-based methods for motion planning, reachable set computations for linear systems, and graph search for finite discrete systems. Our approach does not necessarily require discretization of the system as commonly done in the literature, and is simple to implement in a parallel manner. We demonstrate our approach on Mixed Logical Dynamical (piecewise affine) systems and give simulation results for robotic motion planning problems.

I. INTRODUCTION

We consider the problem of automatically creating control policies for discrete-time nonlinear hybrid systems that are constrained by temporal logic specifications. We are motivated by safety-critical applications in robotics, e.g., autonomous driving and air traffic management, that have non-trivial dynamics and system behaviors (e.g. safety, response, persistence, recurrence, and guarantee) that can be represented by temporal logic. Our approach uses a coarse approximation of the system along with the logical specification to guide the computation constrained reachability problems only as needed to create a control policy.

Feasible control policies can be automatically created for a variety of dynamical systems and temporal logic specifications [1]–[3]. Figure 1) shows an example of what such a control policy and specification might look like. These methods require the construction of a finite discrete abstraction of the dynamical system. An abstraction of a system is a partition of the continuous state space into a finite number of abstract states, i.e., sets of system (concrete) states. Finite abstractions are typically expensive to compute and conservative (see [1]–[7]).

Instead of blindly doing expensive reachability computations to construct an abstraction of a system, we guide reachability computations on the system according to the task specification. This general idea was first introduced as counterexample-guided abstraction refinement (CEGAR) in [8] and extended to hybrid systems in [9], [10]. An abstract model of the system is first created and checked to see if the specification holds. If the check fails, then the abstraction is refined based on a counterexample generated during the check. This counterexample corresponds to a feasible control policy in our context. Lazy abstraction [11] optimizes this process by abstracting the system at different levels. Supervisory controllers for hybrid systems are created in [12]. Our approach is different in that we associate weights with the abstraction and use this to update a ranking of promising discrete plans.

Our work is also related to that on combining task and motion planning [1], [13]–[18]. These approaches first compute a discrete plan and then use sampling-based motion planning techniques to determine if this plan is feasible for the system dynamics. This idea is applied to finite-time LTL properties in [1], [15], [16], [19]. Our work is different in that we consider a wider range of methods of computing reachable sets than sampling-based motion planning.

Finally, coarse bisimulations of discrete-time piecewise-affine systems based on temporal logic specifications are computed in [20]. This work is also similar to CEGAR, but directly computes a formula-equivalent bisimulation. Our approach is different as it is focused on controller synthesis and does not require the exact computation of reachable sets for the system.

We first create an existential abstraction, which is a finite abstraction of the system where transitions between abstract states are assumed to exist, but have yet not been verified to exist in the (concrete) system, e.g., through reachability computations. We then compute an automaton for the LTL formula and create the product automaton. The product au-
maps each state to the set of atomic propositions that are of atomic propositions. The labeling function \( f \) with state \( x \in X \). We will consider a special case of this framework, we create control policies for Mixed Logical Dynamical (MLD) systems \([27]\), which model linear autonomous systems for which one can solve constrained reachability problems, which we then check. This is a general approach that can be used for systems for which one can solve constrained reachability problems. It is also easy to parallelize. As a novel application, we use linear temporal logic (LTL) to concisely and naturally specify desired system behaviors such as response, liveness, safety, stability, priority, and guarantee \([28]\). However, instead of defining the syntax and semantics of LTL, we consider non-deterministic Büchi automata (hereafter called Büchi automata), which accept \( \omega \)-regular languages. Thus, our results hold for any property that can be specified as an \( \omega \)-regular language, which is a regular language extended by infinite repetition (denoted by \( \omega \)). In particular, LTL is a subset of \( \omega \)-regular languages, and an equivalent Büchi automaton can be constructed for any LTL formula \( \varphi \) \([28]\). Figure 2 shows an example Büchi automaton.

**Definition 1.** A Büchi automaton is a tuple \( A = (Q, \Sigma, \delta, Q_0, F) \) consisting of (i) a finite set of states \( Q \), (ii) a finite alphabet \( \Sigma \), (iii) a transition relation \( \delta \subseteq Q \times \Sigma \times Q \), (iv) a set of initial states \( Q_0 \subseteq Q \), (v) and a set of accepting states \( F \subseteq Q \).

Let \( \Sigma^w \) be the set of infinite words over \( \Sigma \). A run for \( \sigma = \Sigma_0 \Sigma_1 \Sigma_2 \ldots \in \Sigma^w \) denotes an infinite sequence \( q_0 q_1 q_2 \ldots \) of states in \( Q \) such that \( q_0 \in Q_0 \) and \( (q_i, \Sigma_i, q_{i+1}) \in \delta \) for \( i \geq 0 \). Run \( q_0 q_1 q_2 \ldots \) is accepting (accepted) if \( q_i \in F \) for infinitely many indices \( i \in \mathbb{N} \) appearing in the run.

Intuitively, a run is accepted by a Büchi automaton if a state in \( F \) is visited infinitely often.

We use the definition of an accepting run in a Büchi automaton and the fact that every LTL formula \( \varphi \) can be
represented by an equivalent Büchi automaton $A_\varphi$ to define satisfaction of an LTL formula $\varphi$.

**Definition 2.** Let $A_\varphi$ be a Büchi automaton corresponding to the LTL formula $\varphi$. A run $\sigma = x_0x_1x_2 \ldots$ in $M$ satisfies $\varphi$, denoted by $\sigma \models \varphi$, if and only if the word $L(\sigma)$ is accepted by $A_\varphi$.

We will often consider a Büchi automaton as a graph with the natural bijection between the states and transitions of the Büchi automaton and the vertices and edges of the graph. Let $G = (S, R)$ be a directed graph with vertices $S$ and edges $R$. There exists an edge $e$ from vertex $s$ to vertex $t$ if and only if $t \in \delta(s, a)$ for some $a \in \Sigma$. A walk $w$ is a finite edge sequence $w = e_0e_1 \ldots e_p$. A cycle is a walk where $e_0 = e_p$.

**C. Set-to-set constrained reachability**

We now define the set-to-set constrained reachability problem, which is a key component of our solution approach. We consider control policies (which compute inputs based on the current state and perhaps a finite-memory) as state feedback is typically needed when exogenous disturbances are present. However, in our examples in Section VI-B, our control policies will simply be open-loop trajectories.

**Definition 3.** Consider a concrete system $M$ of the form (1) where $X_0, X_1, X_2 \subseteq \mathcal{X}$, a non-negative integer horizon length $N$, and a control policy $\mu$ are all given. Set $X_2$ is constrained reachable (under the control policy $\mu$) from set $X_0$, denoted by $X_0 \leadsto X_1, X_2$, if $x(0) \in X_0, x(1), x(2), \ldots, x(N-1) \in X_1, x(N) \in X_2$, and $x(i+1) = f(x(i), \mu(x(i)), d(i))$ for $i = 0, 1, \ldots, N - 1$.

Constrained reachability problem: Given a system $M$ of the form (1) and sets $X_0, X_1, X_2 \subseteq \mathcal{X}$, find a control policy $\mu$ and a non-negative integer horizon length $N$ such that $X_0 \leadsto X_1, X_2$. Return $\mu$ if it exists.

Solving a constrained reachability problem is generally undecidable [4]. However, there exist numerous sound algorithms that compute solutions that under-approximate the true reachable set. Sampling-based algorithms build an approximation of the reachable set and are probabilistically or resolution complete [22]. Optimization-based methods are used for state constrained trajectory generation for nonlinear [23], [24] and linear [25], [29] systems. Computationally expensive PDE-based methods are generally applicable [26]. Finally, for a discrete transition system, computing constrained reachability is simply graph search [30].

We make the standing assumption that there exists a black-box method for computing an under-approximation to a constrained reachability problem for $M$. We denote this method by $\text{CSTREACH}(X_0, X_1, X_2, N, T)$, with initial set $X_0$, constraint set $X_1$, reach set $X_2$, horizon length $N \in \mathbb{N}$, and computation time (or iteration limit) $T \in \mathbb{N}$. For a given query, $\text{CSTREACH}$ returns $\text{Yes}, \text{No}, \text{or TimeOut}$. $\text{Yes}$ means that a control policy exists, which is returned. $\text{No}$ means that a control policy does not exist, and $\text{TimeOut}$ indicates that a control policy was not found in the time limit $T$.

**Algorithm 1**

<table>
<thead>
<tr>
<th>CSTREACH($X_0, X_1, X_2, N, T$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> System $M$, sets $X_0, X_1, X_2 \subseteq \mathcal{X}$, values $N, T \in \mathbb{N}$</td>
</tr>
<tr>
<td><strong>Output:</strong> $\text{Yes}$ and control policy $\mu$, $\text{No}$, $\text{TimeOut}$</td>
</tr>
</tbody>
</table>

**D. Problem statement**

We now formally state the main problem of the paper and give an overview of our solution approach.

**Problem 1.** Given a dynamical system $M$ of the form (1) with initial state $x_0 \in \mathcal{X}$ and an LTL formula $\varphi$, determine whether there exists a control policy $\mu$ such that $M^\mu(x_0) = \varphi$. Return the control policy $\mu$ if it exists.

Problem 1 is undecidable in general due to the continuous dynamics [4]. Thus, we consider sound, but not complete, solutions. Our approach is to create an existential finite abstraction $T$ of the system $M$, without checking reachability between states in $T$. Then, we create a product automaton by combining $T$ with a Büchi automaton $A_\varphi$, representing $\varphi$. An accepting run in the product automaton is an abstract plan.

However, an abstract plan may be infeasible due to dynamic constraints. We check a sequence constrained reachability problems corresponding to an abstract plan. If a control policy is not found, we update the product automaton and search for a new abstract plan. This process is repeated until a feasible trajectory is found, or no more abstract plans exist.

**III. COMBINING THE SYSTEM AND SPECIFICATION**

We now describe an existential finite abstraction $T$. This abstract model over-approximates reachability of the concrete system $M$. The abstract model is easy to compute, but might produce behaviors that the concrete system cannot execute. We then combine this abstract model of the system with an automaton representation of the system.

**A. Existential abstraction**

We use a transition system to represent the existential abstraction of a concrete system $M$ of the form (1).

**Definition 4.** A deterministic (finite) transition system is a tuple $T = (S, R, s_0, AP, L)$ consisting of a finite set of states $S$, a transition relation $R \subseteq S \times S$, an initial state $s_0 \in S$, a set of atomic propositions $AP$, and a labeling function $L : S \rightarrow 2^{2^{AP}}$.

We use the transition system model to define an existential abstraction $T$ for the concrete system $M$ as follows. We partition the concrete system’s state space into equivalence classes of states and associate an abstract state $s \in S$ with each equivalence class. The concretization map $C : S \rightarrow \mathcal{X}$ maps each abstract state to a subset of the concrete system’s state space. A partition is proposition preserving if for every abstract state $s \in S$ and every atomic proposition $p \in AP$, $u \models p$ if and only if $v \models p$ for all concrete states $u, v \in C(s)$ [4]. We do not require our abstraction $T$ to be proposition preserving, which necessitates the non-standard definition of the labeling function.
The abstraction \( \mathcal{T} \) is existential in the sense that there is an abstract transition \((s,t) \in R\) if there exists a control policy that takes the system from some concrete state in \(C(s)\) to some concrete state in \(C(t)\) in finite time. Thus, the existential abstraction \( \mathcal{T} \) is an over-approximation of \( \mathcal{M} \) in the sense that it contains more behaviors, i.e., a series of transitions might exist for the abstraction that does not exist for the concrete system.

We will consider varying levels of abstraction with: a single state, a state for each atomic proposition, a state for each polytope in a polytopic partition of the state space, and a state for a given set of discrete points. A natural question is when to use a fine or coarse abstraction. A coarser abstraction requires solving fewer but potentially challenging constrained reachability problems, while a fine abstraction requires solving a large number of relatively simpler constrained reachability problems. Additionally, it may be easier to compose solutions to constrained reachability problems on a fine abstraction compared to a coarse abstraction, as the size of initial and final sets are smaller. Selecting the appropriate level of abstraction is directly related to the difficulty of solving constrained reachability problems of different sizes.

B. Product automaton

We use a slight modification of the product automaton construction, due to Vardi and Wolper [31], to represent runs that are allowed by the transition system and satisfy the LTL specification.

**Definition 5.** Let \( \mathcal{T} = (S, R, \text{AP}, L) \) be a transition system and \( \mathcal{A} = (Q, 2^\text{AP}, \delta, Q_0, F) \) be a Büchi automaton.

The **weighted product automaton** \( \mathcal{P} = \mathcal{T} \times \mathcal{A} \) is the tuple \( \mathcal{P} = (S_P, \delta_P, F_P, s_{P,0}, \text{AP}_P, L_P, w_P) \), consisting of:

(i) a finite set of states \( S_P = S \times Q \),

(ii) a transition relation \( \delta_P \subseteq S_P \times S_P \), where \( ((s, q), (s', q')) \in \delta_P \) if and only if \((s, s') \in R\) and there exists an \( L \in L(s) \) such that \((q, L, q') \in \delta\),

(iii) a set of accepting states \( F_P = S \times F \),

(iv) a set of initial states \( s_{P,0} \), with \((s_0, q_0) \in S_P, 0 \) if \( q_0 \in Q_0 \),

(v) a set of atomic propositions \( \text{AP}_P = Q \),

(vi) a labeling function \( L_P : S \times Q \rightarrow 2^Q \), and

(vii) a non-negative valued weight function \( w_P : \delta_P \rightarrow \mathbb{R} \).

A run \( \sigma_P = (s_0, q_0)(s_1, q_1) \ldots \) is accepting if \((s_i, q_i) \in F_P\) for infinitely many indices \( i \in \mathbb{N} \). The projection of a run \( \sigma_P = (s_0, q_0)(s_1, q_1) \ldots \) in the product automaton \( \mathcal{P} \) is the run \( \sigma = s_0s_1 \ldots \) in the transition system. The projection of a finite-memory run in \( \mathcal{P} \) is a finite-memory run in \( \mathcal{T} \) [28].

It is well-known that if there exists an accepting run in \( \mathcal{P} \) for an LTL formula \( \varphi \), then there exists an accepting lasso of the form \( \sigma_P = \sigma_{\text{pre}}(\sigma_{\text{af}})^\omega \), where \( \sigma_{\text{pre}} \) be a finite walk in \( \mathcal{P} \) and \( \sigma_{\text{af}} \) be a finite cycle in \( \mathcal{P} \) [28]. For an accepting run \( \sigma_P \), the suffix \( \sigma_{\text{af}} \) is a cycle in the product automaton \( \mathcal{P} \) that satisfies the acceptance condition, i.e., it includes an accepting state. The prefix \( \sigma_{\text{pre}} \) is a finite run from an initial state \( s_{P,0} \) to a state on an accepting cycle. We call this an abstract lasso (or abstract plan).

The concrete plan is the set of constrained reachability problems corresponding to the transitions along an abstract lasso. Each transition \(((s, q), (s', q')) \in \delta_P\) encodes a constrained reachability problem (see Section II-C) for the concrete system. We enforce that the system remains in \((s, q)\) until it eventually reaches \((s', q')\). Let \( L_1 \in L(s) \) correspond to the set of atomic propositions so that \((q, L_1, q) \in \delta\), and \( L_2 \in L(s) \) correspond to the set of atomic propositions so that \((q, L_2, q') \in \delta\). Let \( X_0 = C(s), X_1 = [[L_1]] \) if there exists the transition \(((s, q), (s', q')) \in \delta_P\) or else \( X_1 = \emptyset \), and \( X_2 = C(s') \cap [[L_2]] \). Then, the existence of a concrete transition corresponding to the abstract transition \(((s, q), (s', q'))\) can be checked by solving \( \text{CSTREACH}(X_0, X_1, X_2, N, T) \), for a given horizon length and time (or iteration) limit. These \( \text{CSTREACH} \) problems are concatenated along the abstract lasso in the obvious manner. Note that an additional loop closure constraint must be added for the cyclic behavior.

We use a weight \( w_P \) on all transitions in \( \delta_P \) to represent the likelihood that the corresponding abstract transition in \( \mathcal{P} \) corresponds to a concrete transition, i.e., that \( \text{CSTREACH} \) returns a feasible control policy. For example, this could be the expected necessary horizon length for the \( \text{CSTREACH} \) problem or the size of the corresponding constraint sets. Using these weights contrasts with most methods (with notable exceptions [1], [16]), which perform expensive reachability computations ahead of time to ensure that all transitions in the product automaton can be executed by the concrete system.

IV. SOLUTION OVERVIEW

We now overview our solution approach as detailed in Algorithm 2. First, create an existential abstraction \( \mathcal{T} \) of the concrete system \( \mathcal{M} \) as described in Section III-A. A Büchi automaton \( \mathcal{A}_\varphi \) representing the LTL formula \( \varphi \) can be computed using standard software [32]. Then, construct the product automaton \( \mathcal{P} = \mathcal{T} \times \mathcal{A}_\varphi \).

The problem is now to find an abstract lasso in \( \mathcal{P} \) that is implementable by the concrete system. Compute a minimal weight abstract lasso, e.g., using Dijkstra’s algorithm. As there are an exponential number of paths in \( \mathcal{P} \), it is important to only select the most promising lassos, thus the heuristic weights on transitions in \( \mathcal{P} \). Given an abstract lasso, it must be checked with respect to the system dynamics. Each abstract lasso corresponds to a sequence of constrained reachability problems. If the concrete plan is feasible, then we have a control policy and are done. If a path is infeasible, then we mark the path as infeasible and update the weights in \( \mathcal{P} \). A simple approach is to increase the weight of each edge along the infeasible path by a constant. Additionally, one may compute constrained reachability along a subpath of the infeasible path in an attempt to determine a specific transition that is the cause. There might not be a single transition that invalidates a path, though. Invalidated paths are stored in a set (checkedPaths) so that they are not repeated. We then
compute another lasso until we find a feasible control policy or have tried every path in $\mathcal{P}$.

A benefit of this simple approach is that it is easy to parallelize. A central process can search for abstract lassos in the product automaton and then worker processes can check constrained reachability on them. The workers report their results to the master, which then modifies its search accordingly. There are interesting tradeoffs between searching for accepting lassos and checking individual transitions in $\mathcal{P}$. This is the subject of future work.

Algorithm 2 Solution overview

**Input:** Dynamical system $\mathcal{M}$, LTL formula $\varphi$, iterLimit $\in \mathbb{N}$

**Output:** Feasible control policy $\mu$

1. Compute existential abstraction $\mathcal{T}$ of system $\mathcal{M}$
2. Compute Büchi automaton $A_\varphi$ from LTL formula $\varphi$
3. Create product automaton $\mathcal{P} = \mathcal{T} \times A_\varphi$
4. Assign heuristic weights on transitions of $\mathcal{P}$
5. checkedPaths $= \emptyset$; iter $= 0$
6. while $\text{iter} < \text{iterLimit}$ do
7.   $\text{iter} + = 1$
8.   Compute $\sigma_\mathcal{P} = \sigma_{\mathcal{P}}(\sigma_{\mathcal{A}})^{\omega}$, the current minimum weight abstract lasso not in checkedPaths
9.   Check constrained reachability problem $\text{CStReach}$ corresponding to $\mathcal{P}$
10. if $\text{CStReach}$ returns $\text{YES}$ then
11.     return control policy $\mu$
12. else
13.     Add lasso $\sigma_\mathcal{P}$ to checkedPaths
14.     (Optional) Check CStReach of sub-paths $\sigma_\mathcal{P}$
15.     Increase weights on transitions along $\sigma_\mathcal{P}$
16. end if
17. end while

We now discuss some tradeoffs between different levels of abstraction. Contrary to the counterexample-guided abstraction refinement framework [10], we assume that the number of states in the abstraction is fixed. Instead, we use information from constrained reachability computations to update a ranking over abstract lassos.

The absolute coarsest abstraction of $\mathcal{M}$ contains only a single state with a self transition. It is labeled with every label that corresponds to a concrete state. This effectively means that the product automaton is the Büchi automaton. This is conceptually appealing as it imposes no discretization of the system. This results in a minimal number of abstract lassos that must be checked by constrained reachability. A predicate abstraction is the next coarsest abstraction. This abstraction maps every set of atomic propositions to an abstract state [9]. A polytopic abstraction assumes that the state space has been partitioned into polytopes. The finest level is when a set of concrete states are abstract states, as in [33] and sampling-based methods [1], [15], [34].

The above discussion can be viewed as a continuous refinement of abstractions that depend on the largest volume of the state space corresponding to an abstract state. Intuitively, it should become easier to concatenate solutions of constrained reachability problems as the volume of state space corresponding to each abstract state shrinks. In the limit, when each abstract state maps to a single concrete state, all constrained reachability problems can be concatenated.

This continuum of abstractions leads to a novel way of thinking about abstraction refinement. First consider an abstraction where each abstract state maps to a single concrete state. If a feasible solution cannot be found on this abstraction, one can iteratively expand the regions around the concrete states until a feasible control policy is found. This is in contrast to typical counterexample-guided abstraction refinement approaches [10] since the abstraction becomes coarser instead of finer.

V. COMPLEXITY

Let $|\varphi|$ be the length of formula $\varphi$. A Büchi automaton $A_\varphi$ representing the LTL formula $\varphi$ has worst-case size $O(2^{\mid \varphi \mid})$ [28]. Given an existential abstraction $\mathcal{T}$ with state set $S$, the product automaton $\mathcal{P}$ has $O(|S|2^{\mid \varphi \mid})$ states. There may be an exponential number of accepting lassos in $\mathcal{P}$ that must be checked via constrained reachability computations. Unfortunately as these are existential set-to-set computations (33), it is not guaranteed that solutions to individual problems can be patched together. This contrasts with $(\forall \exists)$ computations [7]. The complexity of checking a constrained reachability problem depends on the system under consideration.

**Proposition 1.** Algorithm 2 is complete in the sense that it will eventually return every accepting lasso in $\mathcal{P}$.

As there are an exponential number of such lassos, and completeness is mostly a theoretical curiosity. Our approach depends on having a good guide to find solution, as evidenced by accurate heuristic weights on the product automaton transitions.

VI. AN APPLICATION TO MIXED LOGICAL DYNAMICAL SYSTEMS

A. Mixed Logical Dynamical systems

As an application, we consider Mixed Logical Dynamical (MLD) systems, which are formally equivalent to piecewise affine systems [27]. These discrete-time systems have both continuous and discrete-valued states and allow one to model nonlinearities, logic, and constraints. Following [35], an MLD system is given by

$$x(t + 1) = Ax(t) + B_1 u(t) + B_2 \delta(t) + B_3 z(t)$$

$$y(t) = Cx(t) + D_1 u(t) + D_2 \delta(t) + D_3 z(t)$$

subject to $E_2 \delta(t) + E_3 z(t) \leq E_1 u(t) + E_4 x(t) + E_5$, (2)

where $t = 0, 1, 2, \ldots$, $x \in \mathbb{R}^{n_x} \times \{0, 1\}^{n_l}$ are the continuous and binary states, $u \in \mathbb{R}^{n_u} \times \{0, 1\}^{m_1}$ are the inputs, $y \in \mathbb{R}^{n_y} \times \{0, 1\}^{n_2}$ are the outputs, and $\delta \in \{0, 1\}^{n_3}$, $z \in \mathbb{R}^{n_4}$ are auxiliary binary and continuous variables, respectively. The terms $A, B_1, B_2, B_3, C, D_1, D_2, D_3, E_1, E_2, E_3, E_4$, and $E_5$ are system matrices of appropriate dimension.
For notational convenience, let $\mathcal{X} = \mathbb{R}^{n_x} \times \{0,1 \}^{n_l}$ and $\mathcal{U} = \mathbb{R}^{n_u} \times \{0,1 \}^{n_l}$. Let $\{ \mathcal{P} \} = \{ x \in \mathcal{X} \mid H_i^p x \leq h^p_i \text{ for some } i \in \mathcal{I}^p \}$ denote the set of states where atomic proposition $p \in AP$ is True. This set is the finite union of polyhedrons (finite conjunctions of halfspaces), which may be non-convex.

We assume that the MLD system (2) is well-posed (see (35)). Thus, for an initial condition $x_0$ and a control input sequence $u = u_0u_1u_2 \ldots$, there is a unique run $\sigma = x_0x_1x_2 \ldots$ that satisfies the constraints in (2). A control input sequence takes the place of a control policy here.

There are various techniques for computing approximate solutions to the constrained reachability problem (see Section II-C) for MLD systems [27], [36]. As this problem is undecidable [4], we impose a finite horizon length $N$ to use mixed-integer optimization methods. The mixed-integer formulation handles integer variables in the dynamics and also non-convex state constraints, e.g., unions of polyhedrons. However, one can still capture infinite behavior by repeating a cyclic finite trajectory.

One can specify a fixed horizon length between each set of constrained reachability problems, or can leave the horizon length as a free variable. Additionally, one can decompose the problem by first computing an accepting loop and then computing a prefix that reaches this loop from the initial state, instead of computing both simultaneously. In both cases, the former approach is computationally more efficient, but can miss feasible solutions. One approach is trying a simpler method first and then switching to more expensive methods as needed.

B. Examples

The following examples demonstrate our methods for tasks motivated by robot motion planning in a planar environment (see Figure 3). All computations were done on a laptop with a 2.4 GHz dual-core processor and 4 GB of memory using CPLEX [37] with Yalmip [38].

We consider the discrete-time constrained linear system

\[
\begin{align*}
x_1(t+1) &= x_1(t) + x_3(t) + 0.5u_1(t), \\
x_2(t+1) &= x_2(t) + x_4(t) + 0.5u_2(t), \\
x_3(t+1) &= x_3(t) + u_1(t), \\
x_4(t+1) &= x_4(t) + u_2(t),
\end{align*}
\]

with $|u_1| \leq 1$, $|u_2| \leq 1$, $|x_3| \leq 1$, and $|x_4| \leq 1$. This planar system is a discrete-time double integrator in orthogonal directions. The variables $x_1, x_2$ are position and $x_3, x_4$ are velocity.

The robot described by these dynamics operates in a 5 x 5 grid environment (see Figure 3). The robot must stay in the safe region $S$ and visit different subsets of the atomic propositions $A, B, C, D$. We consider task specifications of the form $\text{specF}(1) = \bigwedge_{i=1}^n F_{i_1} \land \Box S$ and $\text{specGF}(n) = \bigwedge_{i=1}^n G_{i_1} \land \Box S$, where $F_1 = A$, $F_2 = B$, $F_3 = C$, and $F_4 = D$. We used a fixed horizon $N = 20$ and a time limit $T = 90$ for each constrained reachability problem.

We use the coarsest possible abstraction of the dynamical system; a single abstract state as described in Section IV. This abstraction is not proposition-preserving and effectively means that we use only the Büchi automaton representing the LTL specification to guide the constrained reachability problems that we solve. This is different than previous approaches that build a fine abstraction that is proposition-preserving. Results are shown in Figure 4, where computation times are averaged over five arbitrary (feasible) problems. It never required solving more than two constrained reachability problems to determine a feasible solution. This is likely because the ordering between visits to the different atomic propositions did not affect the feasibility of the solution. The intuition here is that the robot can move to any state in the safe region $S$. 
VII. CONCLUSIONS

We created control policies for discrete-time nonlinear hybrid systems with temporal logic specifications. Our approach uses a coarse approximation of the system along with the logical specification to guide the computation constrained reachability problems only as needed to create a control policy. Notably, we do not require any discretization of the original system and the method lends itself to a parallel implementation.

There are multiple directions for future work, including investigating tradeoffs between checking an entire sequence of constrained reachability problems vs. only a subsequence, investigating tradeoffs between checking an entire sequence of constrained reachability problems only as needed to create a control implementation. Notably, we do not require any discretization of the system along with the logical specification to guide the computation constrained reachability problems.

ACKNOWLEDGEMENTS

This work was supported by a NDSEG fellowship, the Boeing Corporation, and AFOSR award FA9550-12-1-0302.

REFERENCES