Automaton-Guided Controller Synthesis for Nonlinear Systems with Temporal Logic

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

***Abstract*— We develop a method for the control of discrete-time nonlinear systems subject to temporal logic specifications. Our approach uses a coarse abstraction of the system and an automaton representing the temporal logic specification to guide the search for a feasible trajectory. This decomposes the search for a feasible trajectory into a series of constrained reachability problems. Thus, one can create controllers for any system for which techniques exist to compute (approximate) solutions to constrained reachability problems. Representative techniques include sampling-based methods for motion planning, reachable set computations for linear systems, and graph search for finite discrete systems. Our approach avoids the expensive computation of a discrete abstraction, and its implementation is amenable to parallel computing. We demonstrate our approach with numerical experiments on temporal logic motion planning problems with high-dimensional (10+ states) continuous systems.**

I. Introduction

We consider the problem of automatically synthesizing controllers for discrete-time nonlinear systems with temporal logic task specifications. We are motivated by safety-critical applications involving autonomous ground and air vehicles carrying out complex tasks. These systems have nonlinear dynamics and require behaviors (e.g. safety, response, persistence, recurrence, and guarantee) that can be specified with temporal logic. These behaviors generalize traditional point-to-point motion planning [1]. Figure 1 shows an example temporal logic motion planning problem.

Common approaches to temporal logic motion planning construct a finite discrete abstraction of the dynamical system [2]–[4]. 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 states, with transitions between abstract states that represent possible system behaviors. Finite abstractions are typically expensive to compute, conservative, and not guided by the underlying specification (see [2]–[8]).

Instead of blindly doing expensive reachability computations to construct an abstraction of a dynamical system, we use a coarse abstraction of the system and perform constrained reachability checks as needed for the task. 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, e.g., through reachability computations, to exist in the system. We then create a product automaton from the finite-state abstraction and an automaton representing the underlying specification. The product automaton guides reasoning about complicated temporal logic properties as a sequence of simple temporal properties that can be analyzed using constrained reachability techniques. This sequence of constrained reachability problems is called an abstract plan. However, the system might not be able to follow a given abstract plan since dynamic constraints were not considered in the existential abstraction. Thus, we check the abstract plan with the continuous dynamics by solving a sequence of constrained reachability problems. If this sequence is infeasible, the product automaton is updated and a new abstract plan is generated.

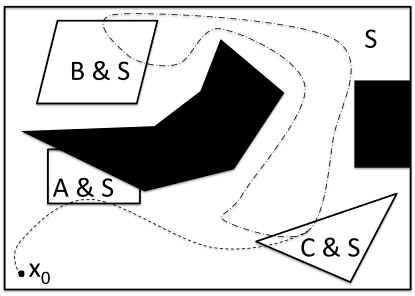

Fig. 1. Sketch of a system trajectory that satisfies the temporal logic specification $\varphi = \Diamond A \wedge \Box \Diamond B \wedge \Box \Diamond C \wedge \Box S$.

This approach lets one take advantage of the significant work in computing constrained reachability relations over continuous state spaces. The related literature includes robotic motion planning [1], optimization-based methods for trajectory generation [9]–[11], and PDE-based methods [12]. Exactness in computing constrained reachability is not critical; we will only require a sound technique.

Our main contribution is a general framework for synthesizing controllers for nonlinear dynamical systems subject to temporal logic specifications. Our approach is independent of the specific techniques used to compute constrained reachability, and is amenable to parallel computing. Despite the framework's generality, it is also computationally powerful, as we show with examples that improve on state-of-the-art techniques for temporal logic motion planning for high-dimensional (10+ states) continuous systems.

Our work is part of the counterexample-guided abstraction refinement (CEGAR) framework [13]–[15]. Here, an abstract model of the system is checked to see if the specification holds. If the check fails, then the abstraction is refined based on a counterexample (e.g. a system trajectory) generated during the check. Our approach differs in that we associate

Eric M. Wolff and Richard M. Murray are with the Department of Control and Dynamical Systems, California Institute of Technology, Pasadena, CA. Ufuk Topcu is with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA. The corresponding author is ewolff@caltech.edu

weights with the abstraction and use them to update a ranking of promising abstract plans.

Our work is also related to that on combining task and motion planning [2], [16]–[21]. These approaches first compute an abstract plan and then use sampling-based motion planning techniques to check if the plan satisfies the dynamic constraints. This idea is applied to co-safe linear temporal logic specifications in [2], [18], [19]. Our approach is agnostic to the method used to check constrained reachability and applies to a wider class of specifications.

Our approach is also related to the specification-guided work in [22] where they compute feedback controllers. We consider more general systems and specifications. Finally, coarse bisimulations of discrete-time piecewise-affine systems based on temporal logic specifications are computed in [23]. Our approach focuses on controller synthesis and does not require the exact computation of reachable sets for the system.

II. Preliminaries

In this section we give background on the system model and specification language. An *atomic proposition* is a statement that is either $True$ or $False$. The cardinality of a set X is denoted by $|X|$.

A. System model

We consider discrete-time nonlinear systems of the form

$$x_{t+1} = f(x_t, u_t), \quad t = 0, 1, \ldots, \tag{1}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^{n_c} \times \{0,1\}^{n_l}$ are the continuous and binary states, $u \in \mathcal{U} \subseteq \mathbb{R}^{m_c} \times \{0,1\}^{m_l}$ are the inputs, and $x_0 \in \mathcal{X}$ is the initial state. The system is called the *concrete* system to distinguish it from its abstraction, which will be introduced in Section III-A.

Let AP be a finite set of atomic propositions. The *labeling function* $L : \mathcal{X} \to 2^{AP}$ maps the continuous part of each state to the set of atomic propositions that are $True$. The set of states where atomic proposition p holds is denoted by $[\![p]\!]$.

A *run* (*trajectory*) $x = x_0 x_1 x_2 \ldots$ of system (1) is an infinite sequence of its states, where $x_t \in \mathcal{X}$ is the state of the system at index t and for each $t = 0, 1, \ldots$, there exists a control input $u_t \in \mathcal{U}$ such that $x_{t+1} = f(x_t, u_t)$. A *word* is an infinite sequence of labels $L(x) = L(x_0)L(x_1)L(x_2)\ldots$ where $x = x_0 x_1 x_2 \ldots$ is a run. Given an initial state x_0 and a control input sequence u, the resulting run $x(x_0, u)$ is unique.

B. Specification language

We use temporal logic to concisely and unambiguously specify desired system behaviors such as response, liveness, safety, stability, priority, and guarantee [24]. We consider ω-regular languages, which are regular languages extended by infinite repetition (denoted by ω). All ω-regular languages are accepted by non-deterministic Büchi automata (hereafter called Büchi automata). Figure 2 shows an example Büchi automaton.

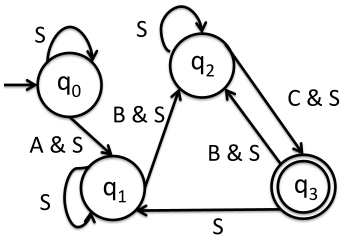

Fig. 2. A (simplified) Büchi automaton corresponding to the LTL formula $\varphi = \Diamond A \wedge \Box\Diamond B \wedge \Box\Diamond C \wedge \Box S$ (stated without definition). Informally, the system must visit A, repeatedly visit B and C, and always remain in S. Here $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{A, B, C, S\}$, $Q_0 = \{q_0\}$, $F = \{q_3\}$, and transitions are represented by labeled arrows.

Definition 1. A Büchi automaton is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ consisting of (i) a finite set of states Q, (ii) a finite alphabet Σ, (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 Σ^ω be the set of infinite words over Σ. A run for $\sigma = \Sigma_0\Sigma_1\Sigma_2 \ldots \in \Sigma^\omega$ denotes an infinite sequence $q_0 q_1 q_2 \ldots$ of states in $\mathcal{A}$ 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 an accepting state, i.e., a state in F, is visited infinitely often.

We will use linear temporal logic (LTL) to specify tasks in our examples. LTL is an ω-regular language, and every LTL formula φ can be automatically translated into a corresponding Büchi automaton $\mathcal{A}_\varphi$ [24].

C. 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 of the form (1) with initial state $x_0 \in \mathcal{X}$ and a Büchi automaton $\mathcal{A}$ representing the specification, determine whether there exists a control input sequence u such that the word $L(x(x_0, u))$ is accepted by $\mathcal{A}$. Return the control u if it exists.

Problem 1 is undecidable in general due to the dynamics over a continuous state space [5]. Thus, we consider sound, but not complete, solutions. Our approach is to create an existential finite abstraction $\mathcal{T}$ of the system that does not necessarily check constrained reachability between states in $\mathcal{T}$. Then, we create a product automaton by combining $\mathcal{T}$ with a Büchi automaton $\mathcal{A}$. An accepting run in the product automaton is an abstract plan. However, an abstract plan may be infeasible due to dynamic constraints. We check the corresponding sequence of constrained reachability problems to see if it is dynamically feasible (see Section IV). If a trajectory 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.

Remark 1. Our general automaton-guided approach may be extended to feedback control of systems with disturbances either by 1) assuming that the disturbances do not change the word, i.e., the sequence of labels, or 2) using an appropriate deterministic automaton.

III. The abstract model

We now describe an *existential* finite abstraction $\mathcal{T}$. This abstract model over-approximates reachability of the concrete system, i.e., it might produce behaviors that the concrete system cannot execute. We then combine this abstract model of the system with the Büchi automaton representing the specification.

A. Existential abstraction

We use a transition system to represent the existential abstraction of a concrete system.

Definition 2. A *deterministic (finite) transition system* is a tuple $\mathcal{T} = (S, R, s_0, AP, L)$ 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 $L : S \to 2^{2^{AP}}$.

We use the finite transition system model to define an existential abstraction $\mathcal{T}$ for the concrete system 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* $C : S \to X \subseteq \mathcal{X}$ maps each abstract state to a subset of the concrete system's state space.

The abstraction $\mathcal{T}$ is existential in the sense that there is an abstract transition $(s, t) \in R$ if there exists a control input such that the system evolves 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 the concrete system 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.

Remark 2. A partition is *proposition preserving* if, for every abstract state $s \in S$ and every atomic proposition $p \in AP$, $p \in L(u)$ if and only if $p \in L(v)$ for all concrete states $u, v \in C(s)$ [5]. We do not require that the abstract states used in creating the existential abstraction $\mathcal{T}$ are proposition preserving, which necessitates the non-standard definition of the labeling function.

B. Product automaton

We use a slight modification of the product automaton construction [25] to represent runs that are allowed by the transition system and satisfy the specification.

Definition 3. Let $\mathcal{T} = (S, R, s_0, AP, L)$ be a transition system and $\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ be a Büchi automaton. The *weighted product automaton* $\mathcal{P} = \mathcal{T} \times \mathcal{A}$ is the tuple $\mathcal{P} = (S_\mathcal{P}, \delta_\mathcal{P}, F_\mathcal{P}, s_{\mathcal{P},0}, AP_\mathcal{P}, L_\mathcal{P}, w_\mathcal{P})$, consisting of

(i) a finite set of states $S_\mathcal{P} = S \times Q$,
(ii) a transition relation $\delta_\mathcal{P} \subseteq S_\mathcal{P} \times S_\mathcal{P}$, where $((s, q), (s', q')) \in \delta_\mathcal{P}$ 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 $F_\mathcal{P} = S \times F$,
(iv) a set of initial states $S_{\mathcal{P},0}$, with $(s_0, q_0) \in S_{\mathcal{P},0}$ if $q_0 \in Q_0$,
(v) a set of atomic propositions $AP_\mathcal{P} = Q$,
(vi) a labeling function $L_\mathcal{P} : S \times Q \to 2^Q$, and
(vii) a non-negative valued weight function $w_\mathcal{P} : \delta_\mathcal{P} \to \mathbb{R}$.

A run $\sigma_\mathcal{P} = (s_0, q_0)(s_1, q_1) \dots$ is *accepting* if $(s_i, q_i) \in F_\mathcal{P}$ for infinitely many indices $i \in \mathbb{N}$. The *projection* of a run $\sigma_\mathcal{P} = (s_0, q_0)(s_1, q_1) \dots$ in the product automaton $\mathcal{P}$ is the run $\sigma = s_0 s_1 \dots$ in the transition system $\mathcal{T}$.

We will often consider an automaton as a graph with the natural bijection between the states and transitions of the 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 $w = e_0 e_1 \dots e_p$. A *cycle* is a walk where $e_0 = e_p$.

It is well-known that if there exists an accepting run in $\mathcal{P}$, then there exists an accepting run of the form $\sigma_\mathcal{P} = \sigma_{\text{pre}}(\sigma_{\text{suf}})^\omega$, where σ_{pre} is a finite walk in $\mathcal{P}$ and σ_{suf} is a finite cycle in $\mathcal{P}$ [24]. For an accepting run $\sigma_\mathcal{P}$, the suffix σ_{suf} is a cycle in the product automaton $\mathcal{P}$ that satisfies the acceptance condition, i.e., it includes an accepting state. The prefix σ_{pre} is a finite run from an initial state $s_{\mathcal{P},0}$ to a state on an accepting cycle. We call $\sigma_\mathcal{P}$ an *abstract plan*.

IV. Concretizing an abstract plan

Given an abstract plan, it is necessary to check if it is feasible for the concrete system. We first define the constrained reachability problem, and then show how to compose these problems to check the feasibility of an abstract plan.

A. Set-to-set constrained reachability

We now define the *set-to-set constrained reachability problem*, which is a key component of our solution approach.

Definition 4. Consider a concrete system of the form (1) with given sets $X_1, X_2 \subseteq \mathcal{X}$, a non-negative integer horizon length N, and a control input sequence u. Set X_2 is *constrained reachable* (under control u) through set X_1, denoted by $X_1 \rightsquigarrow_{X_1} X_2$, if there exist $x_1, \dots, x_{N-1} \in X_1$, $x_N \in X_2$ such that $x_{t+1} = f(x_t, u_t)$ for $t = 1, \dots, N-1$.

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

Solving a constrained reachability problem is generally undecidable [5]. However, there exist numerous sound algorithms that compute solutions to constrained reachability problems. Sampling-based algorithms are probabilistically or resolution complete [1]. Optimization-based methods are used for state constrained trajectory generation for nonlinear [9], [10] and linear [11] systems. Computationally expensive PDE-based methods are generally applicable [12]. Finally,

for a discrete transition system, computing constrained reachability is simply graph search [24].

We make the standing assumption that there exists an oracle for computing a sound solution to a constrained reachability problem for the system. We denote this method by CSTREACH(X_1, X_2, N), with constraint set X_1, reach set X_2, and horizon length $N \in \mathbb{N}$. For a given query, CSTREACH returns YES or NO. YES returns a control input, and NO means that a control input does not exist.

Algorithm 1 CSTREACH(X_1, X_2, N)

Input: Sets $X_1, X_2 \subseteq \mathcal{X}$, and horizon $N \in \mathbb{N}$
Output: YES and control input u, NO

Note that the CSTREACH oracle is sound but not complete. If it does not return a solution after a given amount of computation time, nothing about the constrained reachability problem can be inferred: the problem could be infeasible or feasible but require more computation time.

B. Concretization of abstract plans

The concrete plan is the set of constrained reachability problems corresponding to the transitions along an abstract plan $\sigma = \sigma_{\text{pre}}(\sigma_{\text{suf}})^\omega$. Each transition $((s,q),(s',q')) \in \delta_{\mathcal{P}}$ encodes a constrained reachability problem (see Section IV-A) for the concrete system. We enforce that the system remains in state (s,q) until it eventually reaches state (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 $X_1 = [\![L_1]\!]$ if there exists the transition $((s,q),(s,q)) \in \delta_{\mathcal{P}}$ or else $X_1 = \varnothing$, 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 CSTREACH(X_1, X_2, N), for a given horizon length N. These CSTREACH problems are concatenated along the abstract plan in the obvious manner with a loop closure constraint for the repeated suffix.

We demonstrate the concatenation of CSTREACH problems on the example in Figure 1. We assume that there is a single abstract state s in the existential abstraction, and thus consider transitions (q,q') instead of $((s,q),(s',q'))$. An abstract plan is given by $q_0(q_1q_2q_3)^\omega$ where $\sigma_{\text{pre}} = q_0$ and $\sigma_{\text{suf}} = q_1q_2q_3$. Let x_k^{ij} denote the kth continuous state along the transition from state q_i to q_j. The sequence of states for this abstract plan is $x_1^{01}, \ldots, x_N^{01}, x_1^{12}, \ldots, x_N^{12}, x_1^{23}, \ldots, x_N^{23}, x_1^{31}, \ldots, x_N^{31}$, where $x_1^{12} = f(x_N^{31}, u)$ for some $u \in U$ is the loop closure constraint. The corresponding state constraints for the sequence of CSTREACH problems are $x_1^{01}, \ldots, x_{N-1}^{01} \in [\![S]\!]$, $x_N^{01} \in [\![A \wedge S]\!]$, $x_1^{12}, \ldots, x_{N-1}^{12} \in [\![S]\!]$, $x_N^{12} \in [\![B \wedge S]\!]$, $x_1^{23}, \ldots, x_{N-1}^{23} \in [\![S]\!]$, $x_N^{23} \in [\![C \wedge S]\!]$, and $x_1^{31}, \ldots, x_{N-1}^{31} \in [\![\varnothing]\!]$, $x_N^{31} \in [\![S]\!]$.

V. SOLUTION

We outline our solution to Problem 1 and discuss tradeoffs regarding the levels of granularity of the existential abstraction. Note that if the specification is given as an LTL formula φ, a corresponding Büchi automaton $\mathcal{A}$ can be automatically computed using standard software [26].

A. The solution algorithm

We now overview our solution approach as detailed in Algorithm 2. First, create an existential abstraction $\mathcal{T}$ of the concrete system as described in Section III-A. We discuss tradeoffs on this construction in Section V-B. Then, construct the product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{A}$.

The problem is now to find an abstract plan in $\mathcal{P}$ that is implementable by the concrete system. Compute a minimal weight abstract plan in $\mathcal{P}$, 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 plans. We do this with heuristic weights on transitions in $\mathcal{P}$. The weights $w_{\mathcal{P}}$ represent the likelihood that the corresponding abstract transition in $\mathcal{P}$ corresponds to a concrete transition, i.e., that CSTREACH returns a feasible control input. For example, these weights could be the expected necessary horizon length for the CSTREACH problem or the size of the corresponding constraint sets. Using weights contrasts with most methods in the literature (with notable exceptions [2], [19]), which perform expensive reachability computations ahead of time to ensure that all transitions in the product automaton can be executed by the concrete system.

Given an abstract plan, it must be checked with respect to the system dynamics. Each abstract plan corresponds to a sequence of constrained reachability problems as detailed in Section IV. If the concrete plan if feasible, then a control input is returned. Otherwise, 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 so that they are not repeated. The algorithm then computes another abstract plan until either a feasible control input is found or every path in $\mathcal{P}$ is checked.

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

B. Tradeoffs

We now discuss some tradeoffs between different levels of granularity in the existential abstraction. Contrary to the counterexample-guided abstraction refinement framework [27], 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 plans.

Algorithm 2 Solution overview

Input: Dynamical system, Büchi aut. $\mathcal{A}$, pathLimit $\in \mathbb{N}$
Output: Feasible control input u

```
Compute existential abstraction T of concrete system
Create product automaton P = T × A
Assign heuristic weights to transitions of P
checkedPaths = ∅; paths = 0
while paths < pathLimit do
    paths + = 1
    Compute σ_P = σ_pre(σ_suf)^ω, the current minimum weight abstract plan not in checkedPaths
    Check constrained reachability problem CstReach corresponding to abstract plan
    if CstReach returns Yes then
        return control input u
    else
        Add plan σ_P to checkedPaths
        (Optional) Check CstReach of sub-paths σ_P
        Increase weights on transitions along σ_P
    end if
end while
```

We will consider varying levels of abstraction in which there is: a single state, a state for each atomic proposition, a state for each polytope in a polytopic partition of the state space, or a state for a given set of discrete points. A natural question is when to use a fine or coarse abstraction. Informally, a coarse abstraction requires solving a small number of large constrained reachability problems, while a fine abstraction requires solving a large number of small constrained reachability problems. Additionally, it may be easier to compose solutions to constrained reachability problems on a fine abstraction than a coarse abstraction, as the 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.

The coarsest abstraction of the system contains only a single state with a self transition. It is labeled with every label that corresponds to a concrete state. Thus, the product automaton is the Büchi automaton. This case is conceptually appealing as it imposes no discretization of the system, and results in a minimal number of abstract plans 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 [14]. Thus, the abstraction only depends on the system's labels. 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 [28] and sampling-based methods [2], [18], [29].

The above discussion can be viewed as a continuum 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 size 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, as in sampling-based motion planning. If a feasible solution cannot be found with this abstraction, one can iteratively expand each abstract state to include a larger set of concrete states until a feasible control input can be found. In the limit, the abstract states would partition the state space. This is in contrast to typical counterexample-guided abstraction refinement approaches [27] since the abstraction becomes coarser instead of finer.

VI. Complexity

Given an existential abstraction $\mathcal{T}$ with state set S and a Büchi automaton $\mathcal{A}$, the product automaton $\mathcal{P}$ has $O(|S||\mathcal{A}|)$ states. There may be an exponential number of accepting runs (i.e., abstract plans) in $\mathcal{P}$ that must be checked via constrained reachability computations. 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 return every accepting run in $\mathcal{P}$.*

As there may be an exponential number of accepting runs, completeness is mostly a theoretical curiosity. Our approach depends on having good heuristic weights on the product automaton transitions.

Remark 3. A Büchi automaton $\mathcal{A}_\varphi$ representing the LTL formula φ has worst-case size $O(2^{|\varphi|})$ [24].

VII. An application to nonlinear systems in polygonal environments

We now discuss an application of our framework to nonlinear systems with atomic propositions that can be represented as the unions of polyhedra. We will solve the constrained reachability problems using mixed-integer linear programming. Mixed-integer linear constraints let one specify that the system is in a certain non-convex region (union of polyhedra) at each time step.

A. A mixed-integer formulation of constrained reachability

We assume that each propositional formula ψ is represented by a union of polyhedra. The finite index set I^ψ lists the polyhedra where ψ is $True$. The i-th polyhedron is $\{x \in \mathcal{X} \mid H^{\psi_i} x \leq K^{\psi_i}\}$, where $i \in I^\psi$. Thus, the set of states where atomic proposition ψ is $True$ is given by $[\![\psi]\!] = \{x \in \mathcal{X} \mid H^{\psi_i} x \leq K^{\psi_i} \text{ for some } i \in I^\psi\}$. This set is the finite union of polyhedra (finite conjunctions of half-spaces), and it may be non-convex.

For propositional formula ψ and time t, introduce binary variables $z_t^{\psi_i} \in \{0,1\}$ for all $i \in I^\psi$. Let M be a vector of sufficiently large constants. The *big-M* formulation

$$H^{\psi_i} x_t \leq K^{\psi_i} + M(1 - z_t^{\psi_i}), \quad \forall i \in I^\psi$$
$$\sum_{i \in I^\psi} z_t^{\psi_i} = 1$$

enforces the constraint that $x_t \in [\![\psi]\!]$.

The constrained reachability problem $\textsc{CstReach}(\psi_1, \psi_2, N)$ can then be encoded with the big-M formulation so that $x_t \in [\![\psi_1]\!]$ for $t = 1, \ldots, N-1$ and $x_N \in [\![\psi_2]\!]$. One can specify a fixed horizon length, N, for 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.

B. System dynamics

As the mixed-integer linear constraints in Section VII-A are over a sequence of continuous states, they are independent of the specific system dynamics. Thus, this formulation extends to any deterministic nonlinear system that is amenable to finite-dimensional optimization [9], including small-time locally controllable systems [1]. We highlight two other useful classes of nonlinear systems where the dynamics can be encoded using mixed-integer linear constraints.

Mixed logical dynamical systems

Mixed logical dynamical (MLD) systems have both continuous and discrete-valued states and allow one to model nonlinearities, logic, and constraints [30]. They include constrained linear systems, linear hybrid automata, and piecewise affine systems. An MLD system is of the form

$$\begin{aligned} x_{t+1} &= Ax_t + B_1 u_t + B_2 \delta_t + B_3 z_t \\ \text{subject to } \; & E_2 \delta_t + E_3 z_t \leq E_1 u_t + E_4 x_t + E_5, \end{aligned}$$

where $t = 0, 1, \ldots$, $x \in \mathcal{X} \subseteq \mathbb{R}^{n_c} \times \{0,1\}^{n_l}$ are the continuous and binary states, $u \in \mathcal{U} \subseteq \mathbb{R}^{m_c} \times \{0,1\}^{m_l}$ are the inputs, and $\delta \in \{0,1\}^{r_l}$ and $z \in \mathbb{R}^{r_l}$ are auxiliary binary and continuous variables, respectively. The system matrices A, B_1, B_2, B_3, E_1, E_2, E_3, E_4, and E_5 are of appropriate dimension. We assume that the system is deterministic and well-posed (see Definition 1 in [30]).

Differentially flat systems

A system is *differentially flat* if there exists a set of outputs such that all states and control inputs can be determined from these outputs without integration. If a system has states $x \in \mathbb{R}^n$ and control inputs $u \in \mathbb{R}^m$, then it is flat if one can find outputs $y \in \mathbb{R}^m$ of the form $y = y(x, u, \dot{u}, \ldots, u^{(p)})$ such that $x = x(y, \dot{y}, \ldots, y^{(q)})$ and $u = u(y, \dot{y}, \ldots, y^{(q)})$. Thus, one can plan trajectories in output space and then map these to control inputs.

Differentially flat systems may be encoded using mixed integer linear constraints in certain cases, e.g., the flat output is constrained by mixed integer linear constraints. This condition holds for relevant classes of robotic systems, including quadrotors and car-like robots. However, control input constraints are typically non-convex in the flat output. Common approaches to satisfy control constraints are to plan a sufficiently smooth trajectory or slow down along a trajectory [31].

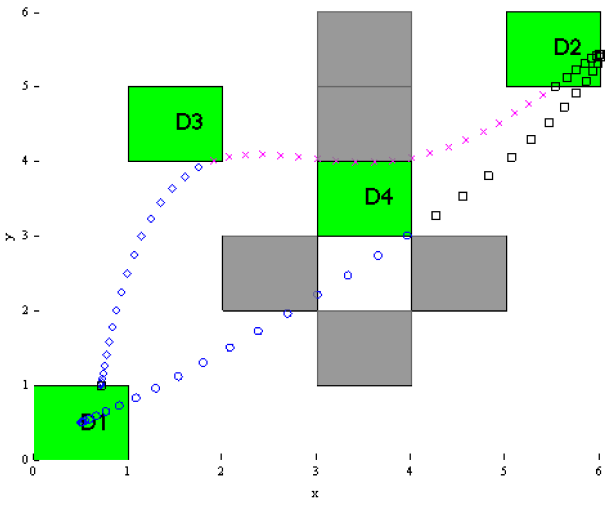

Fig. 3. Illustration of the environment. The goals are labeled $D1$, $D2$, $D3$, and $D4$. Dark regions are obstacles. A representative trajectory for the quadrotor is shown with the five concatenated CSTREACH problems, i.e., $([\![S]\!], [\![D4 \wedge S]\!], 20)$, $([\![S]\!], [\![D2 \wedge S]\!], 20)$, $([\![S]\!], [\![D3 \wedge S]\!], 20)$, $([\![S]\!], [\![D1 \wedge S]\!], 20)$, and $([\![S]\!], [\![S]\!], 20)$ in varied colors and shapes.

C. Computing sets of feasible initial states

Our framework can be extended to compute a *set* of initial states from which there exists a satisfying control input. This is possible (when all labels are unions of polytopes) by performing a projection on a lifted polytope. The key insight is that a satisfying system trajectory has a corresponding sequence of polytopes. One can construct a lifted polytope in the initial state x_0 and control input u, and then project on x_0 to compute a set of feasible initial conditions. We defer to Section V-B in [8] for details on this construction.

VIII. Examples

We demonstrate our techniques on a variety of motion planning problems. The first example is a chain of integrators parameterized by dimension. Our second example is a quadrotor model that was previously considered in [32]. Our final example is a nonlinear car-like vehicle with drift. All computations were done on a laptop with a 2.4 GHz dual-core processor and 4 GB of memory using CPLEX [33] through YALMIP [34].

The environment and task are motivated by a delivery scenario. All properties should be understood to be with respect to regions in the plane (see Figure 3). Let $D1$, $D2$, $D3$, and $D4$ be regions where supplies must be delivered. The robot must stay in the safe region S (in white). Formally, we consider task specifications of the form $\text{specF}(n) = \bigwedge_{i=1}^{n} \Diamond D_i \wedge \Box S$ and $\text{specGF}(n) = \bigwedge_{i=1}^{n} \Box \Diamond D_i \wedge \Box S$ for single and repeated deliveries, respectively.

In the remainder of this section, we consider this temporal logic motion planning problem for different system models. All continuous-time models are discretized using a first-order hold and time-step of 0.5 seconds. We use a fixed horizon $N = 20$ for each constrained reachability problem. These CSTREACH problems are concatenated between two to six

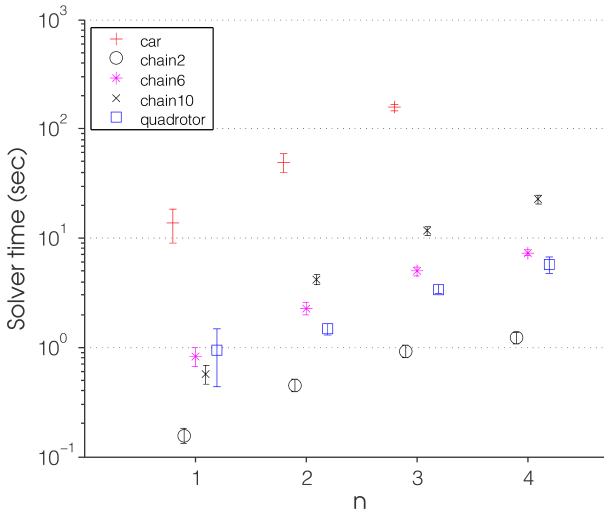

Fig. 4. Solver time (mean ± standard error) to compute a control input for various system models for specF(n).

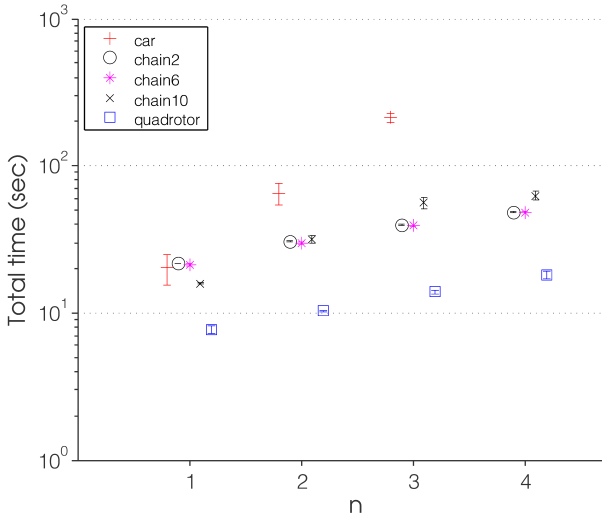

Fig. 5. Total time (mean ± standard error) to compute a control input for various system models for specF(n).

times for an abstract path, resulting in between 40 to 120 time steps (see Figure 3). At each time step, approximately 8 binary variables are used to represent the current label. A computation limit of 60 seconds is enforced for checking reachability of each abstract path, and up to three abstract paths are checked for each trial. Finally, all results are averaged over 20 randomly generated environments.

We use the coarsest possible abstraction of the dynamical system, a single abstract state as described in Section V-B. This abstraction is not proposition-preserving and effectively means that we directly use the Büchi automaton to guide the constrained reachability problems that we solve.

A. *Chain of integrators*

The first system is a chain of orthogonal integrators in the x and y directions. The k-th derivative of the x and y positions are controlled, i.e., $x^{(k)} = u_x$ and $y^{(k)} = u_y$, subject to the constraints $|u_x| \leq 0.5$ and $|u_y| \leq 0.5$. The state constraints are $|x^{(i)}| \leq 1$ and $|y^{(i)}| \leq 1$ for $i = 1, \ldots, k-1$. Results are given in Figures 4, 5, and 6 under "chain-2," "chain-6," and "chain-10," where "chain-k" is a $2k$-dimensional system where the k-th derivative in both the x and y positions is controlled.

B. *Quadrotor*

We now consider the quadrotor model used in [32] for point-to-point motion planning, to which we refer the reader for a complete description of the model. The state $x = (p, v, r, w)$ is 10-dimensional, consisting of position $p \in \mathbb{R}^3$, velocity $v \in \mathbb{R}^3$, orientation $r \in \mathbb{R}^2$, and angular velocity $w \in \mathbb{R}^2$. This model is the linearization of a nonlinear model about hover with the yaw constrained to be zero. The control input $u \in \mathbb{R}^3$ is the total, roll, and pitch thrust. Results are given in Figures 4, 5, and 6 under "quadrotor," and a sample trajectory is shown in Figure 3.

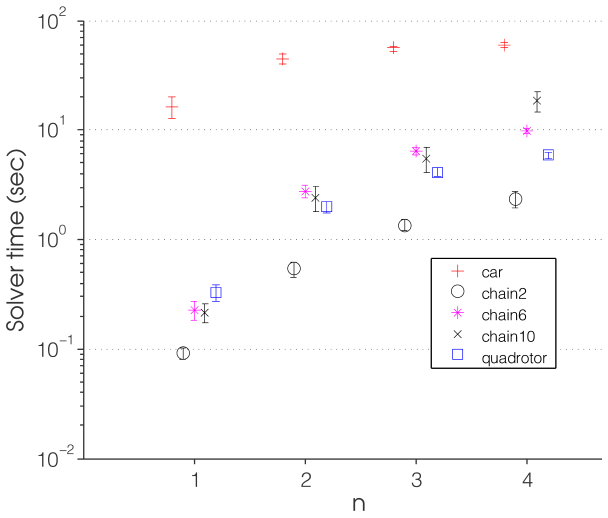

Fig. 6. Solver time (mean ± standard error) to compute a control input for various system models for specGF(n). Total time is not shown.

C. *Nonlinear car*

Consider a nonlinear car-like vehicle with state $x = (p_x, p_y, \theta)$ and dynamics $\dot{x} = (v\cos(\theta), v\sin(\theta), u)$. The variables p_x, p_y are position (m) and θ is orientation (rad). The vehicle's speed v is fixed at 0.5 (m/s) and its control input is constrained as $|u| \leq 2.5$. We form a hybrid MLD model by linearizing the system about different orientations $\hat{\theta}_i$ for $i = 1, 2, 3$. The dynamics are governed by the closest linearization to the current θ. Results are given in Figures 4, 5, and 6 under "car."

D. *Discussion*

We are able to generate satisfying trajectories for 20-dimensional constrained linear systems, which is not possible with finite abstraction approaches such as [4] or [8] or the specification-guided approach of [22]. For the 10-dimensional quadrotor model, feasible solutions are returned

in a matter of seconds. The nonlinear car model required additional binary variables to describe the hybrid modes, which led to larger mixed-integer optimization problems and thus its poor relative performance. Our results appear particularly promising for situations where the environment is dynamically changing and a finite abstraction must be repeatedly computed.

Typically few abstract paths needed to be checked to determine a feasible solution. This is because the ordering between visits to the different labeled regions did not usually affect the problem's feasibility. The intuition is that the robot can (almost) move to any state in the safe region S from any other state.

Finally, the total time (e.g., Figure 4) is typically an order of magnitude more than the solver time (e.g., Figure 5). The main component of the total time is the translation of the YALMIP model to the input for the CPLEX optimizer, which could be avoided by interfacing directly with CPLEX. Thus, we believe that the solver time is more indicative of performance than total time.

IX. Conclusions

We created controllers for discrete-time nonlinear systems with temporal logic specifications. Our approach uses a coarse approximation of the system along with the logical specification to guide the computation of constrained reachability problems as needed for the task. Notably, we do not require any discretization of the original system and our 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, choosing the appropriate abstraction level given a system and a specification, and applying PDE-based methods [12] for the computation of the constrained reachability problems.

Acknowledgements

The authors would like to thank Matanya Horowitz and the anonymous reviewers for their helpful comments. This work was supported by a NDSEG fellowship, the Boeing Corporation, and AFOSR award FA9550-12-1-0302.

References

[1] S. M. LaValle, *Planning Algorithms*. Cambridge Univ. Press, 2006.

[2] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *IEEE Robotics and Automation Magazine*, vol. 18, pp. 55–64, 2011.

[3] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. of IEEE Conf. on Decision and Control*, 2009.

[4] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[5] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proc. IEEE*, vol. 88, no. 7, pp. 971–984, 2000.

[6] C. Belta and L. C. G. J. M. Habets, "Controlling of a class of nonlinear systems on rectangles," *IEEE Trans. on Automatic Control*, vol. 51, pp. 1749–1759, 2006.

[7] L. Habets, P. J. Collins, and J. H. van Schuppen, "Reachability and control synthesis for piecewise-affine hybrid systems on simplices," *IEEE Trans. on Automatic Control*, vol. 51, pp. 938–948, 2006.

[8] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. on Automatic Control*, 2012.

[9] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, 2nd edition*. SIAM, 2000.

[10] M. B. Milam, R. Franz, J. E. Hauser, and R. M. Murray, "Receding horizon control of vectored thrust flight experiment," *IEE Proc., Control Theory Appl.*, vol. 152, pp. 340–348, 2005.

[11] A. G. Richards, T. Schouwenaars, J. P. How, and E. Feron, "Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 25, pp. 755–764, 2002.

[12] C. J. Tomlin, I. M. Mitchell, A. M. Bayen, and M. Oishi, "Computational techniques for the verification of hybrid systems," *Proc. IEEE*, vol. 91, pp. 986–1001, 2003.

[13] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Proc. of CAV*. Springer, 2000.

[14] R. Alur, T. Dang, and F. Ivancic, "Counterexample-guided predicate abstraction of hybrid systems," in *Proc. of TACAS*, 2003.

[15] O. Stursberg, "Synthesis of supervisory controllers for hybrid systems using abstraction refinement," in *Proc. of the 16th IFAC World Congress*, 2005.

[16] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *Int. J. of Robotics Research*, vol. 28, pp. 104–126, 2009.

[17] L. P. Kaelbling and T. Lozano-Perez, "Hierarchical task and motion planning in the now," in *Proc. of IEEE Int. Conf. on Robotics and Automaton*, 2011.

[18] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Trans. on Robotics*, vol. 26, pp. 469–482, 2010.

[19] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *Proc. of IEEE Int. Conf. on Robotics and Automaton*, 2010.

[20] M. P. Vitus, V. Pradeep, J. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Tunnel-MILP: path planning with sequential convex polytopes," in *Proc. of AIAA Guidance, Navigation, and Control Conference*, 2008.

[21] J. Wolfe, B. Marthi, and S. Russell, "Combined task and motion planning for mobile manipulation," in *Proc. of ICAPS*, 2010.

[22] E. A. Gol, M. Lazar, and C. Belta, "Language-guided controller synthesis for discrete-time linear systems," in *Proc. of Hybrid Systems: Computation and Control*, 2012.

[23] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta, "Formal analysis of piecewise affine systems through formula-guided refinement," *Automatica*, vol. 49, pp. 261–266, 2013.

[24] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[25] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *Logic in Computer Science*, 1986, pp. 322–331.

[26] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Proc. of CAV*, 2001.

[27] E. M. Clarke, A. Fehnker, B. H. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald, "Abstraction and counterexample-guided refinement in model checking of hybrid systems," *Int. J. of Foundations of Computer Science*, vol. 14, pp. 583–604, 2003.

[28] J. Liu, U. Topcu, N. Ozay, and R. M. Murray, "Synthesis of reactive control protocols for differentially flat systems," in *Proc. of IEEE Conf. on Decision and Control*, 2012.

[29] S. Karaman, R. G. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," in *Proc. of IEEE Conf. on Decision and Control*, 2008, pp. 2117–2122.

[30] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, pp. 407–427, 1999.

[31] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proc. of Int. Conf. on Robotics and Automation*, 2012.

[32] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2013.

[33] *User's Manual for CPLEX V12.2. IBM, 2010*.

[34] J. Löfberg, "YALMIP : A toolbox for modeling and optimization in MATLAB," in *Proc. of the CACSD Conference*, Taipei, Taiwan, 2004, software available at http://control.ee.ethz.ch/~joloef/yalmip.php.