2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Optimization-based Trajectory Generation with Linear Temporal Logic Specifications

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

***Abstract*— We present a mathematical programming-based method for optimal control of discrete-time dynamical systems subject to temporal logic task specifications. We use linear temporal logic (LTL) to specify a wide range of properties and tasks, such as safety, progress, response, surveillance, repeated assembly, and environmental monitoring. Our method directly encodes an LTL formula as mixed-integer linear constraints on the continuous system variables, avoiding the computationally expensive processes of creating a finite abstraction of the system and a Büchi automaton for the specification. In numerical experiments, we solve temporal logic motion planning tasks for high-dimensional (10+ continuous state) dynamical systems.**

I. Introduction

We are motivated by safety-critical robotics applications involving autonomous air, ground, and space vehicles carrying out complex tasks. In this context, it is important to concisely and unambiguously specify the desired system behavior. Given such as task specification, it is desirable to automatically synthesize a controller that provably implements this behavior. Controller synthesis is complicated by the fact that autonomous systems often have high-dimensional, nonlinear dynamics and require efficient (not just feasible) controllers.

Linear temporal logic (LTL) is an expressive task-specification language for specifying a variety of tasks such as responding to a dynamic environment, visiting goals, periodically monitoring areas, staying safe, and remaining stable. These properties generalize classical point-to-point motion planning. LTL is also promising as a common language for reasoning about the software and dynamics of autonomous systems, due to the widespread use of LTL in software verification [4].

Standard methods for motion planning with LTL task specifications first create a finite abstraction of the original dynamical system (see [2], [5], [15], [23]). This abstraction can informally be viewed as a labeled graph that represents possible behaviors of the system. Given a finite abstraction of a dynamical system and an LTL specification, controllers can be automatically constructed using an automata-based approach [4], [8], [11], [15]. The main drawbacks of this approach are: 1) it is expensive to compute a finite abstraction, and 2) the size of the automaton may be exponential in the length of the specification.

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

Instead of the automata-based approach, we directly encode an LTL formula as mixed-integer linear constraints on the original dynamical system. We enforce that an infinite sequence of system states satisfies the specification by using a finite number of constraints on a trajectory parameterization of bounded length. This is possible by enforcing that the system's trajectory is eventually periodic, i.e., it contains a loop. The loop assumption is motivated by the use of "lassos" in LTL model checking of finite, discrete systems [4]. This direct encoding of the LTL formula avoids the potentially expensive computations of a finite abstraction of the system and a Büchi automaton for the specification.

Our approach applies to any deterministic system model that is amenable to finite-dimensional optimization, as the temporal logic constraints are independent of any particular system dynamics or cost functions. Of particular interest are mixed logical dynamical (MLD) systems [6] and certain differentially flat systems [17], whose dynamics can be encoded with mixed-integer linear constraints. MLD systems include constrained linear systems, linear hybrid automata, and piecewise affine systems. Differentially flat systems include quadrotors [19] and car-like robots.

Our work extends the bounded model checking paradigm for finite, discrete systems [9] to continuous dynamical systems. In bounded model checking, one searches for counterexamples (i.e., trajectories) of a fixed length by transforming the problem into a Boolean satisfiability (SAT) problem. This approach was extended to hybrid systems in [13] by first computing a finite abstraction of the system, and then using a SAT solver for the resulting discrete problem. Hybrid systems are also considered in [3], [12], where SAT solvers are extended to reason about linear inequalities. In contrast, we consider a larger class of dynamics and use mixed-integer linear programming techniques.

Mixed-integer linear programming has been used for trajectory generation for continuous systems with finite-horizon LTL specifications in [14] and [16]. However, finite-horizon properties are too restrictive to model a large class of interesting robotics problems, including surveillance, repeated assembly, periodic motion, and environmental monitoring. A fragment of LTL that includes certain periodic properties is encoded with mixed-integer linear constraints in [22]. We generalize these results by encoding all LTL properties using mixed-integer linear constraints.

Our main contributions are a novel method for encoding LTL specifications as mixed-integer linear constraints on a dynamical system. This generalizes previous Boolean satisfiability encodings of LTL formulas for finite, discrete

978-1-4799-3684-7/14/$31.00 ©2014 IEEE

systems. Our mixed-integer encoding works for any LTL formula, as opposed to previous approaches that only consider finite-horizon properties or fragments. Our encoding is also efficient; it requires a number of variables linear in the length of the trajectory, instead of quadratic as in previous approaches [14]. We demonstrate how our mixed-integer programming formulation can be used with off-the-shelf optimization solvers (e.g. CPLEX [1]) to compute both feasible and optimal controllers for high-dimensional (10+ continuous state) systems with temporal logic specifications.

II. Preliminaries

An *atomic proposition* is a statement that is either $True$ or $False$. A *propositional formula* is composed of only atomic propositions and propositional connectives, i.e., $\wedge$ (and), $\vee$ (or), and $\neg$ (not).

A. System model

We consider discrete-time nonlinear systems of the form

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

where $t = 0, 1, \ldots$ are the time indices, $x \in \mathcal{X} \subseteq (\mathbb{R}^{n_c} \times \{0,1\}^{n_l})$ are the continuous and binary states, $u \in \mathcal{U} \subseteq (\mathbb{R}^{m_c} \times \{0,1\}^{m_l})$ are the control inputs, and $x_0 \in \mathcal{X}$ is the initial state.

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*) $\mathbf{x} = x_0 x_1 x_2 \ldots$ 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 $L(\mathbf{x}) = L(x_0)L(x_1)L(x_2)\ldots$ where $\mathbf{x} = x_0 x_1 x_2 \ldots$ is a run. Given an initial state x_0 and a control input sequence $\mathbf{u}$, the resulting run $\mathbf{x} = \mathbf{x}(x_0, \mathbf{u})$ is unique.

B. Linear temporal logic

We use linear temporal logic (LTL) to concisely and unambiguously specify the desired system behavior [4]. LTL allows the specification of safety, guarantee, liveness, response, and stability properties which generalize traditional point-to-point motion planning [17].

Syntax: LTL is built from (a) a set of atomic propositions AP, (b) Boolean operators: $\neg$ (not), $\wedge$ (and), and $\vee$ (or), and (c) temporal operators: $\bigcirc$ (next), $\mathcal{U}$ (until), and $\mathcal{R}$ (release). An LTL formula in *positive normal form* (negation normal form) is defined by the following grammar:

$$\varphi ::= p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \varphi_1 \,\mathcal{R}\, \varphi_2,$$

where $p \in AP$ is an atomic proposition.

Every LTL formula can be rewritten in positive normal form, where all negations only appear in front of atomic propositions [4]. The following rules transform a given LTL formula into an equivalent LTL formula in positive normal form: $\neg True = False$, $\neg\neg\varphi = \varphi$, $\neg(\varphi_1 \wedge \varphi_2) = \neg\varphi_1 \vee \neg\varphi_2$, $\neg\bigcirc\varphi = \bigcirc\neg\varphi$, and $\neg(\varphi_1 \,\mathcal{U}\, \varphi_2) = \neg\varphi_1 \,\mathcal{R}\, \neg\varphi_2$. An LTL formula φ of size $|\varphi|$ can always be rewritten as a formula φ' in positive normal form of size $|\varphi'| = O(|\varphi|)$ [4].

The standard Boolean operators $\implies$ (implies) and $\iff$ (equivalent) can be defined as $p \implies q = \neg p \vee q$ and $p \iff q = (p \implies q) \wedge (q \implies p)$, respectively. Commonly used abbreviations for LTL formulas are the derived temporal operators $\Diamond\psi = True \,\mathcal{U}\, \psi$ (eventually), $\Box\psi = \neg\Diamond\neg\psi$ (always), $\Box\Diamond\psi$ (always eventually), and $\Diamond\Box\psi$ (eventually always).

Semantics: The semantics of an LTL formula is defined over an infinite sequence of states $\mathbf{x} = x_0 x_1 x_2 \ldots$. Let $\mathbf{x}_i = x_i x_{i+1} x_{i+2} \ldots$ denote the run $\mathbf{x}$ from position i. The semantics are defined inductively as follows:

$$\begin{aligned}
\mathbf{x}_i &\vDash p \text{ if and only if (iff) } p \in L(x_i)\\
\mathbf{x}_i &\vDash \neg p \text{ iff } \mathbf{x}_i \nvDash p\\
\mathbf{x}_i &\vDash \psi_1 \vee \psi_2 \text{ iff } \mathbf{x}_i \vDash \psi_1 \vee \mathbf{x}_i \vDash \psi_2\\
\mathbf{x}_i &\vDash \psi_1 \wedge \psi_2 \text{ iff } \mathbf{x}_i \vDash \psi_1 \wedge \mathbf{x}_i \vDash \psi_2\\
\mathbf{x}_i &\vDash \bigcirc\psi \text{ iff } \mathbf{x}_{i+1} \vDash \psi\\
\mathbf{x}_i &\vDash \psi_1 \,\mathcal{U}\, \psi_2 \text{ iff } \exists j \geq i \text{ s.t. } \mathbf{x}_j \vDash \psi_2 \text{ and } \mathbf{x}_n \vDash \psi_1 \forall i \leq n < j\\
\mathbf{x}_i &\vDash \psi_1 \,\mathcal{R}\, \psi_2 \text{ iff } \forall j \geq i : \mathbf{x}_j \vDash \psi_2 \text{ or } \mathbf{x}_n \vDash \psi_1 \exists i \leq n < j.
\end{aligned}$$

Informally, the notation $\bigcirc\varphi$ means that φ is true at the next step, $\Box\varphi$ means that φ is always true, $\Diamond\varphi$ means that φ is eventually true, $\Box\Diamond\varphi$ means that φ is true infinitely often, and $\Diamond\Box\varphi$ means that φ is eventually always true [4].

Definition 1. A run $\mathbf{x} = x_0 x_1 x_2 \ldots$ *satisfies* φ, denoted by $\mathbf{x} \vDash \varphi$, if $\mathbf{x}_0 \vDash \varphi$.

III. Problem Statement

In this section, we formally state both a feasibility and an optimization problem and give an overview of our solution approach. Let φ be an LTL formula defined over AP.

Problem 1. Given a system of the form (1) and an LTL formula φ, compute a control input sequence $\mathbf{u}$ such that $\mathbf{x}(x_0, \mathbf{u}) \vDash \varphi$.

To distinguish among all trajectories that satisfy Problem 1, we introduce a generic cost function $J(\mathbf{x}, \mathbf{u})$ that maps trajectories and control inputs to $\mathbb{R} \cup \infty$.

Problem 2. Given a system of the form (1) and an LTL formula φ, compute a control input sequence $\mathbf{u}$ such that $\mathbf{x}(x_0, \mathbf{u}) \vDash \varphi$ and $J(\mathbf{x}(x_0, \mathbf{u}), \mathbf{u})$ is minimized.

We now briefly overview our solution approach. We represent the system trajectory as a finite sequence of states. Infinite executions of the system are captured by enforcing that a loop occurs in this sequence, making the trajectory eventually periodic. We then encode an LTL formula as mixed-integer linear constraints on this finite trajectory parameterization in Section IV-B. Additionally, both dynamic constraints (see Section IV-B.3) and a cost function can also be included as part of the mixed-integer optimization problem. For mixed logical dynamical (or piecewise affine) and certain differentially flat systems, Problems 1 and 2 (with

linear costs) can thus be solved using a mixed-integer linear program (MILP) solver. While even checking feasibility of a MILP is NP-hard, modern solvers using branch and bound methods routinely solve large problems [1]. We give results on systems with more than 10 continuous states in Section V.

Remark 1. We only consider open-loop trajectory generation, which is already a challenging problem due to the LTL specifications and continuous dynamics. Incorporating disturbances during trajectory generation is the subject of future work.

IV. Solution

We build on the work in bounded model checking of finite, discrete systems [9], and extend it to dynamical systems using mixed-integer programming. Our presentation and notation follows that of [10].

We will search for a trajectory of length k that satisfies φ. Although a satisfying trajectory for an LTL formula describes an *infinite* sequence of states, an infinite sequence can be captured by a *finite* trajectory that has a loop. Note that this approach is conservative for systems of the form (1) due to both the bounded trajectory length k and the assumption that the trajectory is eventually periodic. This is in contrast to finite systems, where the assumption that the trajectory is eventually periodic is without loss of generality.

Definition 2. A run $\mathbf{x}$ is a (k,l)-*loop* if $\mathbf{x} = (x_0 x_1 \dots x_{l-1})(x_l \dots x_k)^\omega$ such that $0 < l \leq k$ and $x_{l-1} = x_k$, where ω denotes infinite repetition.

Definition 3. Given a run $\mathbf{x}$ and a bound $k \in \mathbb{N}$, $\mathbf{x} \vDash_k \varphi$ iff $\mathbf{x}$ is a (k,l)-loop for some $0 < l \leq k$ and $\mathbf{x}_0 \vDash \varphi$.

For a bounded trajectory length k, Problems 1 and 2, with $\mathbf{x} \vDash \varphi$ replaced by $\mathbf{x} \vDash_k \varphi$, can be represented as a finite-dimensional mixed-integer program. We will build a set $[\![M, \varphi, k]\!]$ of mixed-integer constraints that is satisfiable if and only if a trajectory of length k exists for system M that satisfies φ. The satisfiability of these constraints can then be checked with a mixed-integer programming solver. In the following sections, we will describe the construction of the loop constraints, the LTL constraints, and the system constraints. However, we first detail how to relate the continuous state to the set of valid atomic propositions.

Remark 2. Our results can be extended to the bounded semantics for LTL, i.e., the no-loop case, as detailed in [10].

A. Representing the labels

We now relate the state of a system to the set of atomic propositions that are $True$ at each time instance. We assume that each propositional formula ψ is described by the Boolean combination of a finite number of halfspaces. Our approach here is standard, see e.g., [6], [14].

1) Halfspace representation: We will give a necessary and sufficient condition on a state x_t being in a halfspace $H = \{x \in \mathcal{X} \mid h^T x \leq k\}$ at time t. This can be extended to unions of polyhedra using conjunctions and disjunction as detailed in Section IV-A.2.

We introduce binary variables $z_t \in \{0,1\}$ for time indices $t = 0, \dots, k$. We enforce that $z_t = 1$ if and only if $h^T x_t \leq k$ with the following constraints

$$h^T x_t \leq k + M_t(1 - z_t),$$
$$h^T x_t > k - M_t z_t + \epsilon,$$

where M_t are large positive numbers and ϵ is a small positive number. Note that $z_t = 1$ if and only if the state is in the halfspace H at time t (with precision ϵ on the boundary).

2) Boolean operations: In this section, we encode $\neg$ (not), $\wedge$ (and), and $\vee$ (or) of propositions using mixed-integer linear constraints. We assume that each proposition p has a corresponding variable (binary or continuous) P_t^p which equals 1 if p is $True$ at time t, and equals 0 otherwise. We will use new continuous variables $P_t^\psi \in [0,1]$ to represent the resulting propositions. In each case, $P_t^\psi = 1$ if ψ holds at time t and $P_t^\psi = 0$ otherwise.

The negation of proposition p, i.e., $\psi = \neg p$, is modeled for $t = 0, \dots, k$ as

$$P_t^\psi = 1 - P_t^p.$$

The conjunction of propositions p_i for $i = 1, \dots, m$, i.e., $\psi = \wedge_{i=1}^m p_i$, is modeled for $t = 0, \dots, k$ as

$$P_t^\psi \leq P_t^{p_i}, \quad i = 1, \dots, m,$$
$$P_t^\psi \geq 1 - m + \sum_{i=1}^m P_t^{p_i}$$

The disjunction of propositions p_i for $i = 1, \dots, m$, i.e., $\psi = \vee_{i=1}^m p_i$, is modeled for $t = 0, \dots, k$ as

$$P_t^\psi \geq P_t^{p_i}, \quad i = 1, \dots, m,$$
$$P_t^\psi \leq \sum_{i=1}^m P_t^{p_i}.$$

B. A mixed-integer encoding

We now encode Problem 1 as a set $[\![M, \varphi, k]\!]$ of mixed-integer constraints, which includes loop constraints, LTL constraints, and system constraints. Note that while the loop and LTL constraints are always mixed-integer *linear* constraints, the system constraints will depend on the dynamic model used. Problem 2 uses the same constraint set $[\![M, \varphi, k]\!]$, with the addition of a cost function defined over the (k,l)-loop.

1) Loop constraints: The loop constraints are used to determine where a loop is formed in the system trajectory. We introduce k binary variables $l_1, \dots, l_k$ which determine where the trajectory loops. These are constrained so that only one loop selector variable is allowed to be $True$, and if l_j is $True$, then $x_{j-1} = x_k$. These constraints are enforced by $\sum_{i=1}^k l_i = 1$ and

$$x_k \leq x_{j-1} + M_j(1 - l_j), \quad j = 1, \dots, k,$$
$$x_k \geq x_{j-1} - M_j(1 - l_j), \quad j = 1, \dots, k,$$

where M_j are sufficiently large positive numbers.

2) LTL constraints: Given a formula φ, we denote the satisfaction of φ at position i by $[\![\varphi]\!]_i$. The variable $[\![\varphi]\!]_i \in \{0,1\}$ corresponds to an appropriate set of mixed-integer linear constraints so that $[\![\varphi]\!]_i = 1$ if and only if φ holds at position i. We recursively generate the mixed-integer linear constraints corresponding to $[\![\varphi]\!]_0$ to determine whether or not a formula φ holds in the initial state, i.e., if $[\![\varphi]\!]_0 = 1$. In this section, we use the encoding of the $\mathcal{U}$ and $\mathcal{R}$ temporal operators from [10] for Boolean satisfiability. Our contribution here is linking this satisfiability encoding to the continuous system state through the mixed-integer linear constraints described in Section IV-A. This encoding first computes an under-approximation for $\mathcal{U}$ and an over-approximation for $\mathcal{R}$ with the auxiliary encoding $\langle\!\langle\cdot\rangle\!\rangle$. The under-approximation of $\varphi_1 \,\mathcal{U}\, \varphi_2$ assumes that φ_2 does not hold in the successor of state x_k. The over-approximation of $\varphi_1 \,\mathcal{R}\, \varphi_2$ assumes that φ_2 holds in the successor of state x_k. These approximations are then refined to exact values by $[\![\cdot]\!]$. The encoding is recursively defined over an LTL formula, where there is a case for each logical and temporal connective.

The reader might wonder why an auxiliary encoding is necessary. A seemingly straightforward approach for $\mathcal{U}$ is to use the textbook identity (see [4]) $[\![\psi_1 \,\mathcal{U}\, \psi_2]\!]_i = [\![\psi_2]\!]_i \vee ([\![\psi_1]\!]_i \wedge [\![\psi_1 \,\mathcal{U}\, \psi_2]\!]_{i+1})$ for $i = 0, \ldots, k$, where index $k+1$ is replaced by an appropriate index to form a loop. However, this approach can lead to circular reasoning. Consider a trajectory consisting of a single state with a self loop, and the LTL formula $\mathit{True}\,\mathcal{U}\,\psi$, i.e., $\Diamond\psi$ (eventually). The corresponding encoding is $[\![\mathit{True}\,\mathcal{U}\,\psi]\!]_0 = [\![\psi]\!]_0 \vee [\![\mathit{True}\,\mathcal{U}\,\psi]\!]_0$. This can be trivially satisfied by setting $[\![\mathit{True}\,\mathcal{U}\,\psi]\!]_0$ equal to 1, regardless of whether or not ψ is visited. The auxiliary encoding prevents this circular reasoning, as detailed in [10].

We first define the encoding of propositional formulas as

$$
\begin{aligned}
[\![\psi]\!]_i &= P_i^{\psi}, \\
[\![\neg\psi]\!]_i &= P_i^{\neg\psi}, \\
[\![\psi_1 \wedge \psi_2]\!]_i &= [\![\psi_1]\!]_i \wedge [\![\psi_2]\!]_i, \\
[\![\psi_1 \vee \psi_2]\!]_i &= [\![\psi_1]\!]_i \vee [\![\psi_2]\!]_i,
\end{aligned}
$$

for $i = 0, \ldots, k$, where these operations were defined in Section IV-A using mixed-integer linear constraints.

Next, we define the auxiliary encodings of $\mathcal{U}$ and $\mathcal{R}$. The until (release) formulas at k use the auxiliary encoding $\langle\!\langle\psi_1 \,\mathcal{U}\, \psi_2\rangle\!\rangle_j$ ($\langle\!\langle\psi_1 \,\mathcal{R}\, \psi_2\rangle\!\rangle_j$) at the index j where the loop is formed, i.e., where l_j holds. The auxiliary encoding of the temporal operators is

$$
\begin{aligned}
\langle\!\langle\psi_1 \,\mathcal{U}\, \psi_2\rangle\!\rangle_i &= [\![\psi_2]\!]_i \vee ([\![\psi_1]\!]_i \wedge \langle\!\langle\psi_1 \,\mathcal{U}\, \psi_2\rangle\!\rangle_{i+1}), \\
\langle\!\langle\psi_1 \,\mathcal{R}\, \psi_2\rangle\!\rangle_i &= [\![\psi_2]\!]_i \wedge ([\![\psi_1]\!]_i \vee \langle\!\langle\psi_1 \,\mathcal{R}\, \psi_2\rangle\!\rangle_{i+1}),
\end{aligned}
$$

for $i = 1, \ldots, k-1$, and is

$$
\begin{aligned}
\langle\!\langle\psi_1 \,\mathcal{U}\, \psi_2\rangle\!\rangle_i &= [\![\psi_2]\!]_i, \\
\langle\!\langle\psi_1 \,\mathcal{R}\, \psi_2\rangle\!\rangle_i &= [\![\psi_2]\!]_i,
\end{aligned}
$$

for $i = k$.

Finally, we define the encoding of the temporal operators as

$$
\begin{aligned}
[\![\bigcirc\psi]\!]_i &= [\![\psi]\!]_{i+1}, \\
[\![\psi_1 \,\mathcal{U}\, \psi_2]\!]_i &= [\![\psi_2]\!]_i \vee ([\![\psi_1]\!]_i \wedge [\![\psi_1 \,\mathcal{U}\, \psi_2]\!]_{i+1}), \\
[\![\psi_1 \,\mathcal{R}\, \psi_2]\!]_i &= [\![\psi_2]\!]_i \wedge ([\![\psi_1]\!]_i \vee [\![\psi_1 \,\mathcal{R}\, \psi_2]\!]_{i+1}),
\end{aligned}
$$

for $i = 0, \ldots, k-1$, and as

$$
\begin{aligned}
[\![\bigcirc\psi]\!]_i &= \bigvee_{j=1}^{k} (l_j \wedge [\![\psi]\!]_j), \\
[\![\psi_1 \,\mathcal{U}\, \psi_2]\!]_i &= [\![\psi_2]\!]_i \vee ([\![\psi_1]\!]_i \wedge (\bigvee_{j=1}^{k} (l_j \wedge \langle\!\langle\psi_1 \,\mathcal{U}\, \psi_2\rangle\!\rangle_j))), \\
[\![\psi_1 \,\mathcal{R}\, \psi_2]\!]_i &= [\![\psi_2]\!]_i \wedge ([\![\psi_1]\!]_i \vee (\bigvee_{j=1}^{k} (l_j \wedge \langle\!\langle\psi_1 \,\mathcal{R}\, \psi_2\rangle\!\rangle_j))),
\end{aligned}
$$

for $i = k$.

We also explicitly give the encodings for safety, persistence, and liveness formulas. These formulas frequently appear in specifications and can be encoded more efficiently than the general approach just described.

A safety formula $\Box\psi$ can be encoded as

$$
\begin{aligned}
[\![\Box\psi]\!]_i &= [\![\psi]\!]_i \wedge [\![\Box\psi]\!]_{i+1}, \quad i = 0, \ldots, k-1 \\
[\![\Box\psi]\!]_k &= [\![\psi]\!]_k.
\end{aligned}
$$

An auxiliary encoding is not necessary here, as noted in [10].

Due to the loop structure of the trajectory, both persistence and liveness properties either hold at all indices or no indices. We encode a persistence $\Diamond\Box\psi$ and liveness $\Box\Diamond\psi$ formulas as

$$
\begin{aligned}
[\![\Diamond\Box\psi]\!] &= \bigvee_{i=1}^{k} \left(l_i \wedge \bigwedge_{j=i}^{k} [\![\psi]\!]_j \right), \\
[\![\Box\Diamond\psi]\!] &= \bigvee_{i=1}^{k} \left(l_i \wedge \bigvee_{j=i}^{k} [\![\psi]\!]_j \right),
\end{aligned}
$$

for $i = 0, \ldots, k$. Although the encodings for persistence and liveness appear to require a number of variables that are quadratic in k, one can share subformulas to make this linear in k [9].

3) System constraints: The system constraints encode valid trajectories of length k for a system of the form (1), i.e., they hold if and only if trajectory $\mathbf{x}(x_0, \mathbf{u})$ satisfies the constraints in equation (1) for $t = 0, 1, \ldots, k$.

System constraints, e.g., the dynamics in equation (1), can be encoded on the sequence of states using standard transcription methods [7]. However, the resulting optimization problem may then be a mixed-integer *nonlinear* program due to the dynamics.

A useful class of nonlinear systems where the dynamics can be encoded using mixed-integer *linear* constraints are mixed logical dynamical (MLD) systems. MLD systems can model nonlinearities, logic, and constraints [6]. These include constrained linear systems, linear hybrid automata, and piecewise affine systems.

Additionally, differentially flat systems can sometimes be encoded using mixed-integer linear constraints, e.g., if the

labels over the flat output can be represented as a finite union of polyhedra. However, input constraints are typically non-convex the output space, and thus hard to enforce. Input constraints can often be satisfied by planning smooth trajectories or changing the time-scaling [20]. Many vehicles are differentially flat, including quadrotors and simple car-like robots [17].

The intersection of the LTL constraints, the system constraints, and the loop constraints gives the full mixed-integer linear encoding of the bounded model checking problem, i.e., $[\![M, \varphi, k]\!]$. Problem 1 is solved by checking the feasibility of this set using a MILP solver (assuming the dynamics are mixed logical dynamical). Similarly, Problem 2 is solved by optimizing over this set using a MILP solver (assuming the cost function is linear). More general dynamics and cost functions can be included by using an appropriate mixed-integer solver, potentially at the expense of completeness.

C. Complexity

The main source of complexity of our approach comes from the number of binary variables needed to relate the satisfaction of an atomic proposition to the continuous state of the system. A binary variable is introduced at each time index for each halfspace. If n_h halfspaces are used to represent the atomic propositions used in the LTL formula at each time index, then $O(k \cdot n_h)$ binary variables are introduced, where k is the number of time indices.

Continuous variables are used to represent the propositions introduced during the encoding of the LTL formula. The number of continuous variables used is $O(k \cdot |\varphi|)$, where k is the number of time indices and $|\varphi|$ is the length of the formula. This linear dependence on k improves on the the quadratic dependence on k in [14]. Finally, although the loop constraints introduce k additional binary variables, they are constrained such that only one is active. In summary, our mixed-integer linear encoding of an LTL formula φ requires the use of $O(k \cdot n_h)$ binary variables and $O(k \cdot |\varphi|)$ continuous variables. Note that the complexity of solving a mixed-integer linear program is worst-case exponential in the number of binary variables [1].

We do not give bounds on the number of constraints, as they depend heavily on the details of a given specification, and are not the major factor in the complexity of solving a mixed-integer linear program.

Remark 3. Our solution approaches for Problems 1 and 2 are only complete with respect to a given bound of k. If a solution is not found for a given value of k, there may exist a solution for a larger value. Future work will identify cases when an upper bound on k can be computed a priori, similar to the notion of a completeness threshold [10].

V. Examples

We consider two LTL motion planning problems for different system models. We demonstrate our techniques on a chain of integrators model, a quadrotor model from [21], and a nonlinear car-like vehicle with drift. All computations are done on a laptop with a 2.4 GHz dual-core processor and 4 GB of memory using CPLEX [1] through Yalmip [18]. All continuous-time models are discretized with a 0.35 second sample time.

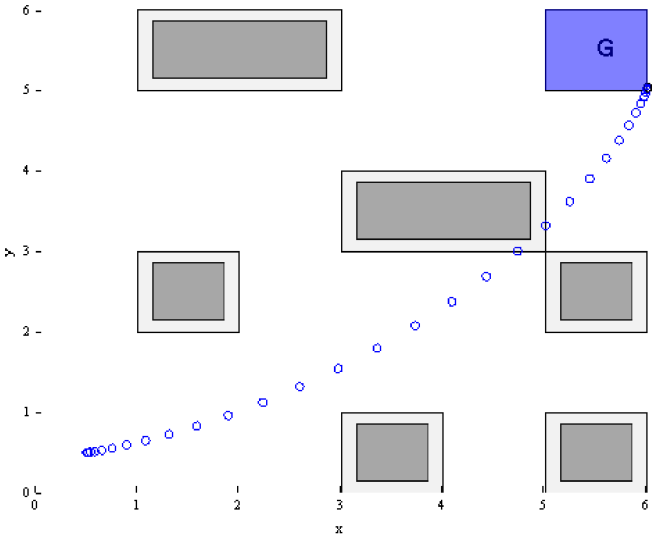

Fig. 1. Illustration of the environment for the reach-avoid scenario. The goal is labeled G and dark regions are obstacles. The light regions around the obstacles give a safety margin so that the continuous evolution of the system does not intersect an obstacle, which would otherwise be possible since constraints are enforced at discrete time steps. A representative trajectory for the "chain-6" model is shown, where we additionally minimized an additive cost function with cost $c(x_t, u_t) = |u_t|$ accrued at each time index.

Our first example is a simple reach-avoid motion planning scenario (see Figure 1), which we use to directly compare our encoding of the until operator ($\mathcal{U}$) with that given in [14]. The task here is to remain in the safe region S until the goal region G is reached. The corresponding LTL formula is $\varphi = S \,\mathcal{U}\, G$. This formula is a key component for building more complex LTL formulas, and thus performance of this encoding is important. We show that our formulation scales better with respect to the trajectory length k. This is anticipated, as our encoding of $\mathcal{U}$ requires a number of variables linear in k, while the encoding in [14] requires a number of variables quadratic in k. For this example, all results are averaged over ten randomly generated environments where 75 percent of the area is safe (labeled S), i.e., the remaining regions are obstacles.

The second example is motivated by a surveillance mission. Let A, B, C, and D describe regions of interest in a planar planning space that must be repeatedly visited (see Figure 2). The robot must remain in the safe region S (in white) and either visit regions A and B repeatedly or visit regions C and D repeatedly. Formally, the task specification is $\varphi = \Box S \wedge ((\Box \Diamond A \wedge \Box \Diamond B) \vee (\Box \Diamond C \wedge \Box \Diamond D))$. This formula is recursively parsed according to the grammar in Section II-B and encoded as mixed-integer linear constraints as described in Section IV-B. For this example, all results are averaged over five randomly generated environments where 85 percent of the area is safe, i.e., the remaining regions are obstacles. The length of the trajectory is $k = 25$, corresponding to 8.75 seconds.

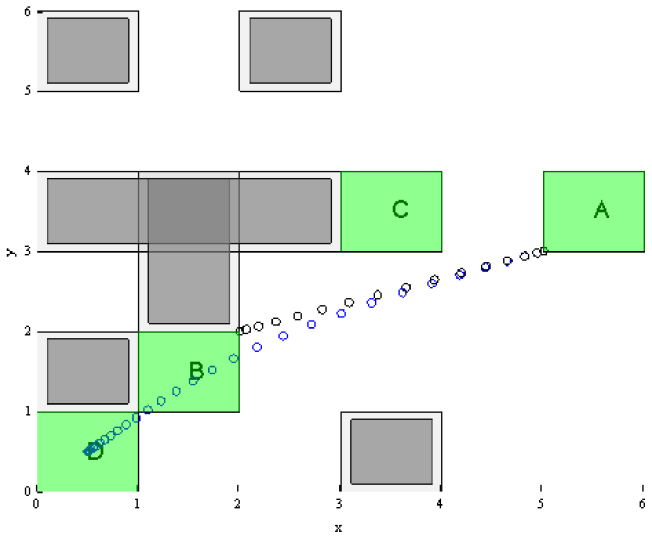

Fig. 2. Illustration of the environment for the surveillance scenario. The goals are labeled A, B, C, and D. Dark regions are obstacles. A representative trajectory for the quadrotor model is shown, where we additionally minimized an additive cost function with cost $c(x_t, u_t) = |u_t|$ accrued at each time index. The system starts in region D and repeatedly visits regions A and C.

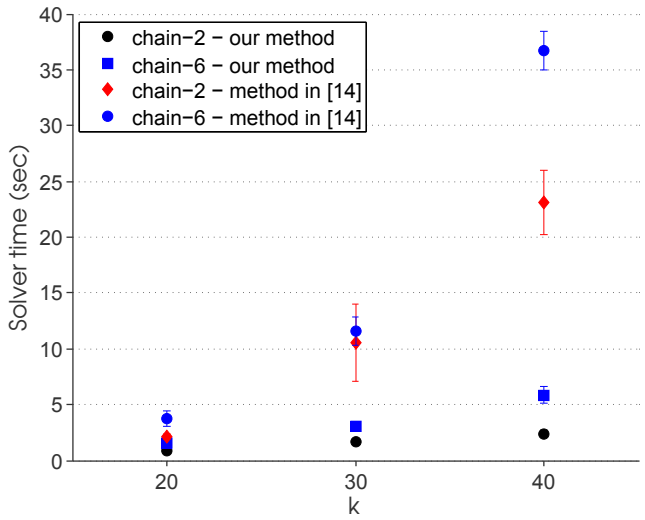

Fig. 3. Solver time (mean ± standard error) to compute a feasible control input for the reach-avoid example with our approach and the approach in [14].

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 $y^{(k)} = u_y$. The control input constraints are $|u_x| \leq 0.5$ 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 Figure 4 under "chain-2," "chain-6," and "chain-10," where "chain-k" indicates that the k-th derivative in both the x and y positions are controlled.

B. Quadrotor

We now consider the quadrotor model used in [21], to which we refer the reader for details. 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, and with the yaw constrained to be zero. The control input $u \in \mathbb{R}^3$ is the total, roll, and pitch thrust. A sample trajectory is shown in Figure 2, and results are given in Figure 4 under "quadrotor."

C. Nonlinear car

Finally, we 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 1 (m/s) and its control input is constrained as $|u| \leq 2.5$.

We form a hybrid mixed logical dynamical model by linearizing the system about different orientations $\hat{\theta}_i$ for $i = 1, \ldots, k$. The dynamics are governed by the closest linearization to the current θ. Results with $k = 4$ are given in Figure 4 under "car-4."

D. Discussion

The results for the first example are presented in Figure 3, where we used the "chain-2" and "chain-6" models. Our encoding of the key $\mathcal{U}$ temporal operator scaled significantly better than a previous approach, likely due to the fact that our encoding is linear in the trajectory length k, while the previous encoding was quadratic in k.

The results for the surveillance example are presented in Figure 4. Due to the periodic nature of these tasks, the approaches presented in [14] and [16] are not applicable. We were able to quickly compute trajectories that satisfied periodic temporal tasks for systems with more than 10 continuous states. The total time was dominated by preprocessing in Yalmip, and we expect that this can be reduced close to the solver time with a dedicated implementation.

Finally, we compared our mixed-integer linear programming approach to a reachability-based algorithm that computes a finite abstraction of a continuous system [24]. This method took 22 seconds to compute a discrete abstraction for a simple kinematic system (i.e., "chain-1"). In contrast, our mixed-integer approach finds solutions to LTL motion planning problems for these systems in a few seconds. Abstraction-based approaches scale poorly as the dimension increases, and cannot currently solve problems with more than a handful of dimensions. Our approach is also promising for situations where the environment is dynamically changing and it would be prohibitively expensive to repeatedly compute a finite abstraction.

VI. Conclusions

We presented a mixed-integer programming-based method for optimal control of nonlinear systems subject to linear temporal logic task specifications. We directly encoded an LTL formula as mixed-integer linear constraints on continuous system variables, which avoiding the computationally

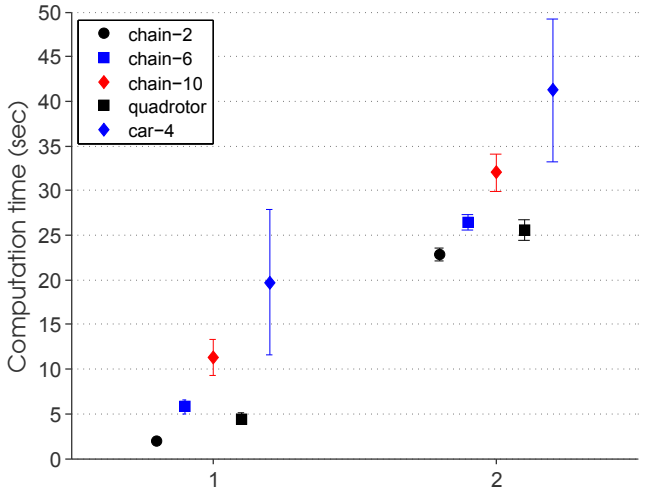

Fig. 4. Solver (1) and total (2) time (mean ± standard error) to compute a feasible control input for various systems for the surveillance example. Each example used an average of 570 binary variables and 4,800 constraints.

expensive processes of creating a finite abstraction of the system and creating a Büchi automaton for the specification. We solved LTL motion planning tasks for dynamical systems with more than 10 continuous states, and showed that our encoding of the until operator theoretically and empirically improves on previous work.

Directions for future work include directly encoding metric and past operators, exploiting incremental computation for varying bounds k, and performing a more detailed comparison of mixed-integer linear programming solvers and Boolean satisfiability solvers that have been extended to support linear operations, e.g., [12]. Reasoning about Boolean operations may be substantially improved using modern SMT solvers.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful feedback. This work was supported by an NDSEG fellowship, the Boeing Corporation, AFOSR award FA9550-12-1-0302, and ONR award N00014-13-1-0778.

References

[1] *User's Manual for CPLEX V12.2. IBM, 2010.*
[2] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proc. IEEE*, 88(7):971–984, 2000.
[3] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying industrial hybrid systems with MathSAT. *Electronic Notes in Theoretical Computer Science*, 119:17–32, 2005.
[4] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
[5] C. Belta and L. C. G. J. M. Habets. Controlling of a class of nonlinear systems on rectangles. *IEEE Trans. on Automatic Control*, 51:1749–1759, 2006.
[6] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35:407–427, 1999.
[7] J. T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, 2nd edition*. SIAM, 2000.
[8] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi. Motion planning with complex goals. *IEEE Robotics and Automation Magazine*, 18:55–64, 2011.
[9] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. of TACAS*, 1999.
[10] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2:1–64, 2006.
[11] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45:343–352, 2009.
[12] M. Fränzle and C. Herde. Efficient proof engines for bounded model checking of hybrid systems. *Electronic Notes in Theoretical Computer Science*, 133:119–137, 2005.
[13] N. Giorgetti, G. J. Pappas, and A. Bemporad. Bounded model checking of hybrid dynamical systems. In *Proc. of IEEE Conf. on Decision and Control*, 2005.
[14] 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*, pages 2117–2122, 2008.
[15] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. on Automatic Control*, 53(1):287–297, 2008.
[16] Y. Kwon and G. Agha. LTLC: Linear temporal logic for control. In *Proc. of HSCC*, pages 316–329, 2008.
[17] S. M. LaValle. *Planning Algorithms*. Cambridge Univ. Press, 2006.
[18] 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.
[19] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *In Proc. of Int. Conf. on Robotics and Automaton*, 2011.
[20] 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.
[21] 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.
[22] E. M. Wolff and R. M. Murray. Optimal control of nonlinear systems with temporal logic specifications. In *Proc. of Int. Symposium on Robotics Research*, 2013. (to appear).
[23] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning. *IEEE Trans. on Automatic Control*, 2012.
[24] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. TuLiP: A software toolbox for receding horizon temporal logic planning. In *Proc. of Int. Conf. on Hybrid Systems: Computation and Control*, 2011. http://tulip-control.sf.net.