Synthesis of Switching Protocols from Temporal Logic Specifications

Jun Liu, Necmiye Ozay, Ufuk Topcu, and Richard M. Murray

Abstract

We propose formal means for synthesizing switching protocols that determine the sequence in which the modes of a switched system are activated to satisfy certain high-level specifications in linear temporal logic. The synthesized protocols are robust against exogenous disturbances on the continuous dynamics. Two types of finite transition systems, namely under- and over-approximations, that abstract the behavior of the underlying continuous dynamics are defined. In particular, we show that the discrete synthesis problem for an under-approximation can be formulated as a model checking problem, whereas that for an over-approximation can be transformed into a two-player game. Both of these formulations are amenable to efficient, off-the-shelf software tools. By construction, existence of a discrete switching strategy for the discrete synthesis problem guarantees the existence of a continuous switching protocol for the continuous synthesis problem, which can be implemented at the continuous level to ensure the correctness of the nonlinear switched system. Moreover, the proposed framework can be straightforwardly extended to accommodate specifications that require reacting to possibly adversarial external events. Finally, these results are illustrated using three examples from different application domains.

Index Terms

Hybrid systems, switching protocols, formal synthesis, linear temporal logic, model checking, temporal logic games.

This work was supported in part by the NSERC of Canada, the FCRP consortium through the Multiscale Systems Center (MuSyC), and the Boeing Corporation.

The authors are with Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125, USA. {liu, necmiye, utopcu, murray}@cds.caltech.edu

I. Introduction

The objective of this paper is synthesizing switching protocols that determine the sequence in which the modes of a switched system are activated to satisfy certain high-level specifications formally stated in linear temporal logic (LTL). Different modes may correspond to, for example, the evolution of the system under different, pre-designed feedback controllers [1], [2], so-called motion primitives in robot motion planning [3], [4], or different configurations of a system (e.g., different gears in a car or aerodynamically different phases of a flight). Each of these modes may meet certain specifications but not necessarily the complete, mission-level specification the system needs to satisfy. The purpose of the switching protocol is to identify a switching sequence such that the resulting switched system satisfies the mission-level specification.

Specifically, given a family of system models, typically given as ordinary differential equations potentially with bounded exogenous disturbances, and an LTL specification, our approach builds on a hierarchical representation of the system in each mode. The continuous evolution is accounted for at the low level. The higher level is composed of a finite-state approximation of the continuous evolution. The switching protocols are synthesized using the high-level, discrete evolution. Simulation-type relations [5] between the continuous and discrete models guarantee that the correctness of the synthesized switching protocols is preserved in the continuous implementation.

We consider two types of finite-state approximations for continuous nonlinear systems, namely under- and over-approximations. Roughly speaking, we call a finite transition system $\mathcal{T}$ an under-approximation if every transition in $\mathcal{T}$ can be continuously implemented for all allowable exogenous disturbances. In the case in which an under-approximation based finite-state abstraction is used, the switching protocol synthesis can be formulated as a model checking [6] problem. On the other hand, a finite transition system $\mathcal{T}$ is called an over-approximation if for each transition in $\mathcal{T}$, there is a possibility (due to either the exogenous disturbances or the coarseness of the approximation) for continuously implementing the strategy. We account for the mismatch between the continuous model and its over-approximation as adversarial uncertainty and model it nondeterministically. Consequently, the corresponding switching protocol synthesis

problem is formulated as a two-player temporal logic game (see [7] and references therein and the pioneering work in [8]). This game formulation also allows us to incorporate adversarial environment variables that do not affect the dynamics of the system but constrain its behavior through the specification.

Fragments of the switching protocol synthesis problem considered here have attracted considerable attention. We now give a very brief overview of some of the existing work as it ties to the proposed methodology (a thorough survey is beyond the scope of this paper). Jha *et al.* [9] focuses on switching logics that guarantee the satisfaction of certain safety and dwell-time requirements. Taly and Tiwari [10], Cámara *et al.* [11], Asarin *et al.* [12], and Koo *et al.* [13] consider a combination of safety and reachability properties. Joint synthesis of switching logics and feedback controllers for stability are studied by Lee and Dullerud [14]. The work by Frazzoli *et al.* [3], [4] on the concatenation of a number of motion primitives from a finite library to satisfy certain reachability properties constitutes an instance of switching protocol synthesis problem. Our work also has strong connections with the automata-based composition of the so-called interfaces that describe the functionality and the constraints on the correct behavior of a system [15].

The main contributions of the current paper are in extending the family of systems and specifications in switching protocol synthesis. The proposed methodology is applicable to a large family of system models potentially with exogenous disturbances along with an expressive specification language (LTL in this case). The use of LTL enables to handle a wide variety of specifications beyond mere safety and reachability, as well as to account for potentially adversarial, a priori unknown environments in which the system operates (and therefore its correctness needs to be interpreted with respect to the allowable environment behaviors). Furthermore, the methodology improves the flexibility of switching protocol synthesis by merging ideas from multiple complementing directions and offering options that trade computational complexity with conservatism (and expressivity). For example, the resulting problem formulation with under-approximations of continuous evolution is amenable to highly-optimized software for model checking [16], [17], yet at the expense of increased conservatism in modeling. On the other hand, over-approximations are potentially easier to establish, yet the resulting formulation is

a two-player temporal logic game (with publicly available solvers [18], [7] that are less evolved compared to the currently available model checkers). Another trade-off is in the family of two-player games considered here. Such games with complete LTL specifications is known to have prohibitively high computational complexity [19]. Therefore, we focus on an expressive fragment of LTL, namely Generalized Reactivity (1), with favorable computational complexity [7].

The remainder of the paper is organized as follows. In the next section, we introduce finite-state approximations for nonlinear systems subject to exogenous disturbances, and formulate the switching synthesis problem under consideration. In Section III, we solve the switching protocol synthesis problem for under- and over-approximations via model checking and a two-player game approach, respectively. In Section IV, we present three illustrative examples to demonstrate our results. A brief discussion on how to obtain under- and over-approximations is included in Appendix A.

II. Preliminaries and Problem Formulation

A. Continuous-time switched systems

Consider a family of nonlinear systems,

$$\dot{x} = f_p(x, d), \quad p \in \mathcal{P}, \tag{1}$$

where $x(t) \in X \subseteq \mathbb{R}^n$ is the state at time t and $d(t) \in D \subseteq \mathbb{R}^d$ is the exogenous disturbance, $\mathcal{P}$ is a finite index set, and $\left\{f_p : p \in \mathcal{P}\right\}$ is a family of nonlinear vector fields satisfying the usual conditions to guarantee the existence and uniqueness of solutions for each of the subsystems in (1). A *switched system* generated by the family (1) can be written as

$$\dot{x} = f_\sigma(x, d), \tag{2}$$

where σ is a switching signal taking values in $\mathcal{P}$. The value of σ at a given time t may depend on t or $x(t)$, or both, or may be generated by using more sophisticated design techniques [2]. We emphasize that, although the above formulation does not explicitly include a control input in its formulation, it can capture different situations where control inputs can be included, e.g., within each mode p, we may either assign a constant valued control input u_p, which can further belong to a finite number of

quantized levels $\left\{u_p^1, u_p^2, \cdots, u_p^{L_p}\right\} \subseteq \mathbb{R}^m$, or choose a feedback controller $u(t) = K_p(x(t))$. Depending on different applications, each mode in (1) may represent, for example, a control component [15], [20], a motion primitive (which belongs to, e.g., a finite library of flight maneuvers [3], [4], or a set of pre-designed behaviors [21]), and, in general, an operating mode of a multi-modal dynamical system [9], [10]. To achieve complex tasks, it is often necessary to compose these basic components. The composition can be enforced at a high-level control layer by implementing a switching protocol for mission-level specification. Designing correct switching protocols, however, can be a challenging issue [9], [12], [13], [15].

The goal of this paper is to propose methods for automatically synthesizing σ such that solutions of the resulting switched system (2) satisfy, by construction, a given linear temporal logic (LTL) specification, for all possible exogenous disturbances. LTL is a rich specification language that can express many desired properties, including safety, reachability, invariance, response, and/or a combination of these [22] (see also [23] for examples).

B. Problem description and solution strategy

Before formally stating the problem, we present a schematic description of the problem and its solution approach. Given a family of system models in (1) and its specification expressed in LTL, we synthesize a switching control protocol that, by construction, guarantees that the system satisfies its specification for all allowable exogenous disturbance. Within the same formulation, we also aim to incorporate environmental adversaries, which do not directly impact the dynamics of the system but constrain its behavior through the specification, and synthesize effective switching controllers for all valid environment behaviors. The solution of this problem enables us to compose available controllers, which are predesigned to meet certain specifications, to achieve a high-level specification, as illustrated in Figure 1.

Based on the continuous-time nonlinear system model (1), our hierarchical approach to the switching synthesis problem consists of two steps.

(i) We first establish finite-state approximations of the family of systems (1), which are a family of finite transition systems that approximate the dynamics in each

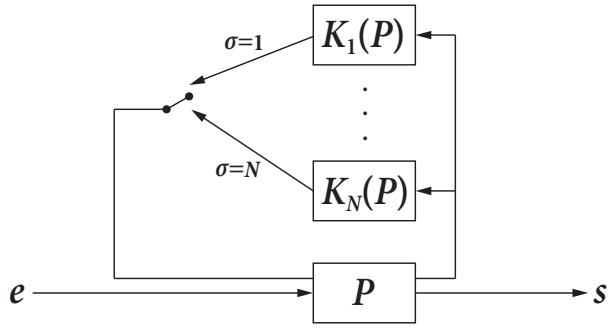

Fig. 1: P represents a plant subject to exogenous disturbances, $\{K_i(P) : i = 1, \cdots, N\}$ is a family of controllers, e represents environmental adversaries, which do not directly impact the dynamics of the system but constrain its behavior through the specification. The objective is to design σ such that the overall system satisfies a high-level specification φ expressed in LTL.

mode.

(ii) We then synthesize a switching protocol based on high-level, discrete abstraction that, when continuously implemented, ensures the correctness of the trajectories of the resulting switched system (2).

More specifically, we formulate two different types of discrete abstractions, namely *under-approximation* and *over-approximation*, respectively. For an under-approximation, the synthesis of a switching protocol is formulated as an LTL model checking [6] problem, which is amenable to highly optimized software implementations [16], [17]. For an over-approximation, we formulate the problem as a two-player temporal logic game. While solving two-player games with general LTL winning conditions is known to have prohibitively high computational complexity [19], we restrict ourselves to an expressive fragment of LTL, namely Generalized Reactivity (1), with favorable computational complexity [7].

While exogenous disturbances are accounted for in the continuous level, adversarial environment behaviors are diverse and not necessarily amenable to modeling as an ordinary differential equation. Therefore, we defer the formal introduction of environment variables to Section III-B, where a two-player game formulation allows us to incorporate

adversarial environment variables that do not affect the continuous-level dynamics of the system but rather constrain its behavior through the high-level specification.

C. Finite-state approximations

To formally state the synthesis problem, we define two types of finite-state abstractions of the continuous evolution in (1) and introduce the specification language LTL. LTL formulas are built upon a finite number of atomic propositions. An *atomic proposition* is a statement on system variables of interest that has a unique truth value (`True` or `False`) for a given value (called *state*) of each system variable. To formulate the switching synthesis problem, we are at least interested in two types of variables: the plant variable x and the switching mode variable p. Let $\Pi := \{\pi_1, \pi_2, \cdots, \pi_n\}$ be a set of atomic propositions. For example, each proposition $\pi_i \in \Pi$ can represent a domain in $\mathbb{R}^n$ and a set of modes in $\mathcal{P}$ of interest. Formally, for system (2), we associate an observation map

$$h : \mathbb{R}^n \times \mathcal{P} \to 2^{\Pi},$$

which maps the continuous states and the discrete modes to a finite set of propositions. Without loss of generality, we consider h to be defined on the whole state space instead of some bounded invariant set. We also allow overlapping set of propositions since h is set-valued instead of single-valued.

Abstractions for each of the subsystems in (1) can be considered by defining an abstraction map $T : \mathbb{R}^n \to Q$, which maps each state $x \in \mathbb{R}^n$ into a finite set $Q := \{q_i : i = 1, \cdots, M\}$. The map T essentially defines a partition of the state space $\mathbb{R}^n$ by $\left\{T^{-1}(q) : q \in Q\right\}$. We shall refer to elements in Q as discrete states of an abstraction. Finite-state approximations are defined in the following.

Definition 1. A *finite transition system* is a tuple $\mathcal{T} := (Q, Q_0, \to)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, and $\to \subseteq Q \times Q$ is a transition relation. Given states $q, q' \in Q$, we write $q \to q'$ if there is a transition from q to q' in $\mathcal{T}$.

Consider a family of finite transition systems

$$\left\{\mathcal{T}_p := (Q, Q_0, \xrightarrow{p}) : p \in \mathcal{P}\right\}. \tag{3}$$

Definition 2. The family of finite transition systems in (3) is said to be an *under-approximation* of (1) if the following two statements hold.

(i) Given states $q, q' \in Q$ such that $q' \neq q$, if there is a transition $q \xrightarrow{p} q'$, then for all $x_0 \in T^{-1}(q)$, there exists some $\tau > 0$ such that, for all exogenous disturbances $d : [0, \tau] \to D \subseteq \mathbb{R}^d$, trajectories ξ of pth subsystem of (1) starting from x_0 satisfy

$$\xi(\tau) \in T^{-1}(q') \quad \xi(t) \in T^{-1}(q) \cup T^{-1}(q'), \quad t \in [0, \tau].$$

(ii) For any $q \in Q$, there is a self-transition $q \xrightarrow{p} q$, then, for all $x_0 \in T^{-1}(q)$ and all exogenous disturbances $d : [0, \infty) \to D \subseteq \mathbb{R}^d$, trajectories ξ of the pth subsystem of (1) starting from x_0 satisfy

$$\xi(t) \in T^{-1}(q), \quad \forall t \in [0, \infty),$$

i.e., $T^{-1}(q)$ is a positively invariant set for the pth subsystem under all exogenous disturbances.

Definition 3. The family of finite transition systems in (3) is said to be an *over-approximation* for (1) if the following two statements hold.

(i) Given states $q, q' \in Q$ such that $q' \neq q$, there is a transition $q \xrightarrow{p} q'$, if there exists $x_0 \in T^{-1}(q)$, $\tau > 0$, and some exogenous disturbance $d : [0, \tau] \to D \subseteq \mathbb{R}^d$ such that the corresponding trajectory ξ of the pth subsystem of (1) starting from x_0, i.e., $\xi : [0, \tau] \to \mathbb{R}^n$ with

$$\xi(0) = x_0, \quad \dot{\xi}(t) = f_p(\xi(t), d(t)), \quad \forall t \in (0, \tau),$$

satisfies

$$\xi(\tau) \in T^{-1}(q') \quad \xi(t) \in T^{-1}(q) \cup T^{-1}(q'), \quad t \in [0, \tau].$$

(ii) For any $q \in Q$, there is a self-transition $q \xrightarrow{p} q$, if there exists $x_0 \in T^{-1}(q)$ and some exogenous disturbance $d : [0, \infty) \to D \subseteq \mathbb{R}^d$ such that the complete trajectory ξ of the pth subsystem of (1) on $[0, \infty)$ starting from x_0 is contained in $T^{-1}(q)$.

Intuitively, in an over-approximation, a discrete transition $q \xrightarrow{p} q'$ is included in $\mathcal{T}_p$ as long as there is a possibility (either induced by disturbances or a coarse partition) for the continuous system to implement the transition, whereas, in an under-approximation,

a discrete transition $q \xrightarrow{p} q'$ is included in $\mathcal{T}_p$ only if the continuous flow can strictly implement the transition. In other words, an under-approximation includes only transitions that can be implemented by the continuous dynamics and an over-approximation includes all possible transitions.

In both approximations, time is abstracted out in the sense that we do not care how much time it takes to reach one discrete state from another. As the focus of this paper is on the automatic synthesis of switching protocols, we shall assume that we are given or we can construct a finite abstraction of the subsystems in (1), which is either an under-approximation or an over-approximation by Definitions2 and 3. A brief discussion on how to obtain such approximations is given in Appendix A (a detailed study is beyond the scope of this paper and subject to current research).

For the above finite approximations to be consistent with continuous dynamics, they should preserve propositions of interest in the sense that for all x, $y \in \mathbb{R}^n$ and $p \in \mathcal{P}$,

$$T(x) = T(y) \Rightarrow h(x,p) = h(y,p), \tag{4}$$

where h is the observation map defined earlier. In other words, if two continuous states belong to the same subset of the continuous state space corresponding to the same discrete state in Q, they should map to the same propositions under h.

The following definitions will be useful when we introduce the semantics of LTL formulas in the next subsection. They are also used later to formally reason about the correctness of the continuous implementations of a synthesized switching protocol.

Definition 4. Given a switching signal $\sigma : \mathbb{R}^+ \to \mathcal{P}$, *trajectories* of the continuous-time switched system (2) are piecewise differentiable functions from $\mathbb{R}^+$ to $\mathbb{R}^n$ that satisfy

$$\dot{x}(t) = f_{\sigma(t)}(x(t), d(t)), \quad t \geq t_0.$$

Without ambiguity, we can write trajectories as pairs $(x(t), \sigma(t))$ or simply (x, σ).

Definition 5. Given a sequence of modes $p_0 p_1 p_2 \cdots$, $p_i \in \mathcal{P}$, $i \geq 0$, a *switching execution* of the family of transition systems in (3) is a sequence of pairs $(q,p) = (q_0,p_0)(q_1,p_1)(q_2,p_2)\cdots$, where $q_0 \in Q_0$, and, for all $i \geq 0$, $q_i \in Q$ and $q_i \xrightarrow{p_i} q_{i+1}$.

D. LTL Syntax and Semantics

We use linear temporal logic (LTL) [24], [22] to formally specify system properties. Standard LTL is built upon a finite set of atomic propositions, logical operators $\neg$ (negation) and $\vee$ (disjunction), and the temporal modal operators $\bigcirc$ (next) and $\mathcal{U}$ (until).

Formally, given a set of atomic propositions Π, the set of LTL formulas over Π can be defined inductively as follows:

(1) any atomic proposition $\pi \in \Pi$ is an LTL formula;

(2) if φ and ψ are LTL formulas, so are $\neg\varphi$, $\bigcirc\varphi$, $\varphi \vee \psi$, and $\varphi \mathcal{U} \psi$.

Additional logical operators, such as $\wedge$ (conjunction), $\rightarrow$ (material implication), and temporal modal operators $\Diamond$ (eventually), and $\Box$ (always), are defined by:

(a) $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$;

(b) $\varphi \rightarrow \psi := \neg\varphi \wedge \psi$;

(c) $\Diamond\varphi :=$ `True` $\mathcal{U}\varphi$;

(d) $\Box\varphi := \neg\Diamond\neg\varphi$.

A *propositional formula* is one that does not include any temporal operators.

In order to reason about the correctness of the trajectories that continuously implement the switching protocol synthesized at the discrete level, we shall interpret LTL over both the continuous trajectories of (2) and discrete executions of (3) given by Definitions 4 and 5.

Continuous Semantics of LTL: An LTL formula for the continuous-time switching system (2) is interpreted over its trajectories (x, σ). Formally, given an LTL formula φ without the next operator $\bigcirc$, we can recursively define the satisfaction of φ over a trajectory $(x(t), \sigma(t))$ at time t, written $(x(t), \sigma(t)) \models \varphi$, as follows:

(1) for any atomic proposition $\pi \in \Pi$, $(x(t), \sigma(t)) \models \pi$ if and only if $\pi \in h(x(t), \sigma(t))$;

(2) $(x(t), \sigma(t)) \models \neg\varphi$ if and only if $(x(t), \sigma(t)) \not\models \varphi$;

(3) $(x(t), \sigma(t)) \models \varphi \vee \psi$ if and only if $(x(t), \sigma(t)) \models \varphi$ or $(x(t), \sigma(t)) \models \psi$; and

(4) $(x(t), \sigma(t)) \models \varphi \mathcal{U} \psi$ if and only if there exists $t' \geq t$ such that $(x(t'), \sigma(t')) \models \psi$ and $(x(s), \sigma(s)) \models \varphi$ for all $s \in [t, t')$.

A trajectory (x, σ) starting at t_0 is said to satisfy φ, written $(x, \sigma) \models_{t_0} \varphi$, if $(x(t_0), \sigma(t_0)) \models \varphi$. If the initial time is not significant, we simply write $(x, \sigma) \models \varphi$.

Remark 1. By allowing the observation map h to be defined on the product space $\mathbb{R}^n \times \mathcal{P}$, we are able to put temporal logic constraints on both the state and the switching mode. Based on the above definition, $\Box\varphi$ holds at time t if and only if φ is satisfied by $(x(t'), \sigma(t'))$ at all future time $t' \geq t$; $\Diamond\varphi$ holds at time t if and only if φ is satisfied by $(x(t'), \sigma(t'))$ holds at some time $t' \geq t$; and the path formula $\varphi \mathcal{U} \psi$ intuitively expresses the property that over trajectory $(x(t), \sigma(t))$, φ is true until ψ becomes true. For example, $(\sigma(t) = p_1)\mathcal{U}(\sigma(t) = p_2)$ expresses that the switched system should stay in mode p_1 until it eventually switches to mode p_2. For $p_1 \neq p_2$, $(\sigma(t) = p_1)\mathcal{U}(\sigma(t) \neq p_2)$ specifies that, once in mode p_1, the system is not allowed to directly switch to mode p_2, which can be useful to rule out unsafe mode transitions. Therefore, we are still able to express sequential properties of the switching modes without using the next operator $\bigcirc$ in the continuous-time setting.

Discrete Semantics of LTL: An LTL formula for a switched system given by the family of transition systems (3) is interpreted over its switching executions. Given an LTL formula φ, we can recursively define the satisfaction of φ over a switching execution $(q, p) = (q_0, p_0)(q_1, p_1)(q_2, p_2)\cdots$ at position i, written $(q_i, p_i) \vDash \varphi$, as follows:

(1) for any atomic proposition $\pi \in \Pi$, $(q_i, p_i) \vDash \pi$ if and only if there exists $x_i \in T^{-1}(q_i)$ such that $\pi \in h(x_i, p_i)$;

(2) $(q_i, p_i) \vDash \neg\varphi$ if and only if $(q_i, p_i) \nvDash \varphi$;

(3) $(q_i, p_i) \vDash \bigcirc\varphi$ if and only if $(q_{i+1}, p_{i+1}) \vDash \varphi$;

(4) $(q_i, p_i) \vDash \varphi \vee \psi$ if and only if $(q_i, p_i) \vDash \varphi$ or $(q_i, p_i) \vDash \psi$;

(5) $(q_i, p_i) \vDash \varphi \mathcal{U} \psi$ if and only if there exists $j \geq i$ such that $(q_j, p_j) \vDash \psi$ and $(q_k, p_k) \vDash \varphi$ for all $k \in [i, j)$.

A switching execution $(q, p) = (q_0, p_0)(q_1, p_1)(q_2, p_2)\cdots$ is said to satisfy φ, written $(q, p) \vDash \varphi$, if $(q_0, p_0) \vDash \varphi$.

Remark 2. In the discrete semantics of LTL, the next operator $\bigcirc$ is allowed, mainly to enforce the discrete transition relations. In addition, as noted in Remark 1, we may be interested in specifying formulas such as $(p = p_1)\mathcal{U}(p \neq p_2)$, where $p_1 \neq p_2$, which excludes unsafe mode transition from mode p_1 to mode p_2. As mentioned earlier, to be able to solve the synthesis problem as a two-player game in polynomial time, we

will restrict our LTL formulas to a subclass called Generalized Reactivity (1), or simply GR(1), formulas. GR(1) formulas, however, do not allow explicit use of the until operator $\mathcal{U}$. The same specification, nevertheless, can be stated using the next operator $\bigcirc$, for example, by $(p = p_1) \rightarrow \bigcirc(p \neq p_2)$. Indeed, we can show that the following two LTL formulas have the same truth value and therefore are equivalent

$$\Box((p = p_1) \rightarrow (p = p_1)\mathcal{U}(p \neq p_2)) \Longleftrightarrow \Box((p = p_1) \rightarrow \bigcirc(p \neq p_2)).$$

E. Relationship between the continuous and discrete LTL semantics

We discuss the relationship between the continuous LTL semantics and discrete LTL semantics as follows.

Definition 6. Given a switching execution $(q, p) = (q_0, p_0)(q_1, p_1)(q_2, p_2)\cdots$, a trajectory (x, σ) is said to be a *continuous implementation* of (q, p) starting from t_0, if there exists a sequence of times $t_0 < t_1 < t_2 < \cdots$ such that

$$x(t) \in T^{-1}(q_k), \quad \sigma(t) = p_k, \quad \forall t \in [t_k, t_{k+1}), \quad \forall k \in \mathbb{Z}^+.$$

We use the notation $(x, \sigma) \preccurlyeq_{t_0} (q, p)$ to represent the above implementation. Furthermore, the implementation is said to be *non-Zeno*, if $t_k \rightarrow \infty$ as $k \rightarrow \infty$.

Proposition 1. Let φ be an LTL formula without the next operator $\bigcirc$. Given a switching execution $(q, p) = (q_0, p_0)(q_1, p_1)(q_2, p_2)\cdots$ and a trajectory (x, σ), starting at t_0, that continuously implements (q, p) in the sense of Definition 6, i.e., $(x, \sigma) \preccurlyeq_{t_0} (q, p)$, then

$$(q, p) \vDash \varphi \quad \text{if and only if} \quad (x, \sigma) \vDash_{t_0} \varphi.$$

Proof: The proposition can be proved recursively. For simplicity, we use $\Longleftrightarrow$ to indicate "if and only".

We first show that the statement of the proposition holds for all atomic propositions. Let $\pi \in \Pi$. By definition, $(q_0, p_0) \vDash \pi$ implies that there exists $x_0 \in T^{-1}(q_0)$ such that $\pi \in h(x_0, p_0)$. Note that $x(t_0) \in T^{-1}(q_0))$ and $\sigma(t_0) = p_0$ by the Definition 6. By (4), we have $h(x(t_0), p_0) = h(x_0, p_0)$ and therefore $\pi \in h(x(t_0), \sigma(t_0))$, which is $(x, \sigma) \vDash \pi$. On the other hand, $\pi \in h(x(t_0), \sigma(t_0))$ implies that there exists $x_0 = x(t_0) \in T^{-1}(q_0)$ such that $\pi \in h(x_0, p_0)$, which shows $(q_0, p_0) \vDash \pi$.

Now suppose that the proposition holds for two LTL formulas φ and ψ, and we want to show it also holds for the formulas $\neg\varphi$, $\varphi \vee \psi$, and $\varphi \mathcal{U} \psi$. Indeed, we have

$$(q_0, p_0) \vDash \neg\varphi \Longleftrightarrow (q_0, p_0) \nvDash \varphi \Longleftrightarrow (x(t_0), \sigma(t_0)) \nvDash \varphi \Longleftrightarrow (x(t_0), \sigma(t_0)) \vDash \neg\varphi,$$

and

$$\begin{aligned}(q_0, p_0) \vDash \varphi \vee \psi &\Longleftrightarrow (q_0, p_0) \vDash \varphi \text{ or } (q_0, p_0) \vDash \psi \\ &\Longleftrightarrow (x(t_0), \sigma(t_0)) \vDash \varphi \text{ or } (x(t_0), \sigma(t_0)) \vDash \psi \Longleftrightarrow (x(t_0), \sigma(t_0)) \vDash \varphi \vee \psi.\end{aligned}$$

To show the equivalence for $\varphi \mathcal{U} \psi$, consider two statements:

(a) there exists some $j \geq 0$ such that $(q_j, p_j) \vDash \psi$ and $(q_k, p_k) \vDash \varphi$ for all $k \in [0, j)$;

(b) there exists some $t \geq t_0$ such that $(x(t), \sigma(t)) \vDash \psi$ and $(x(s), \sigma(s)) \vDash \varphi$ for all $s \in [t_0, t)$.

Here, (a) characterizes $(q_0, p_0) \vDash \varphi \mathcal{U} \psi$ and (b) characterizes $(x(t_0), \sigma(t_0)) \vDash \varphi \mathcal{U} \psi$. We first suppose that (a) holds for some $j \geq 0$ and claim that (b) holds for $t = t_j$. To this end, observe that for each $s \in [t_0, t]$, $(q_{k_s}, p_{k_s}) = (T(x(s)), \sigma(s))$ corresponds to one pair in the finite sequence $(q_0, p_0)(q_1, p_1) \cdots (q_j, p_j)$. Moreover, by definition, $(x(s), \sigma(s))$ is an implementation of the execution $(q_{k_s}, p_{k_s})(q_{k_s+1}, p_{k_s+1}) \cdots$, starting at time s. Therefore, (b) must be true by the recursive assumption. On the other hand, if (b) holds for some t, we can pick $j \geq 0$ such that $t \in [t_j, t_{j+1})$ and show that (a) holds for this j in a similar way.

Therefore, we have shown the equivalence for all LTL formulas without the next operator. ■

F. Problem Formulation

Now we are ready to formally state our switching synthesis problems.

Continuous Switching Synthesis Problem: Given a family of continuous-time subsystems in (1) and a specification φ, synthesize a switching strategy that generates only correct trajectories (x, σ) in the sense that $(x, \sigma) \vDash \varphi$.

Discrete Switching Synthesis Problem: Given a family of finite transition systems in (3) and a specification φ, synthesize a switching strategy that generates only correct switching executions (q, p) in the sense that $(q, p) \vDash \varphi$.

We focus on the discrete synthesis problem and propose two different approaches depending on the types of abstractions in the sense of Definitions 2 and 3. It will be shown that, by construction, our solutions to the discrete switching synthesis problems from both approaches can be continuously implemented to generate a solution for the continuous switching synthesis problem.

III. Synthesis of Switching Protocols

In this section, we propose two approaches, one for each of the two types of finite-state approximations, to the discrete synthesis problem formulated in the previous section.

A. Switching synthesis by model checking

We start with the synthesis of switching protocol for an under-approximation of (1). Given such a finite approximation, the discrete synthesis problem can be reformulated as a model checking problem. Model checking [6], [25] is an automated verification technique that, given a finite-state model of a system and a formal specification, systematically checks whether this specification is satisfied. If not, the model checker provides a counterexample that indicates how the model could violate the specification. This counterexample is usually given as an execution path that that violates the property being verified [6]. This execution path can either be finite, which leads from the initial system state to a single state that violates the property being verified, or be infinite, which leads to a loop of states, which is repeated infinitely many times and violates the property being verified. The counterexample being finite or infinite depends on the property being verified. Roughly speaking, a counterexample for safety and invariant properties is a finite path, while a counterexample for reachability and liveness properties is an infinite execution path [6].

Formally, to solve the switching synthesis problem by model checking, we construct a product transition system $(\mathcal{Q}\times\mathcal{P}, \mathcal{Q}_0\times\mathcal{P}_0, \rightarrow)$ from the family of transition systems $\left\{\mathcal{T}_p\right\}$ in (3). Here $\mathcal{Q}\times\mathcal{P}$ is a set of system states that consist of switching modes $\mathcal{P}$ and plant states $\mathcal{Q}$, $\mathcal{Q}_0\times\mathcal{P}_0$ represents initial states, and $\rightarrow \subseteq (\mathcal{Q}\times\mathcal{P})\times(\mathcal{Q}\times\mathcal{P})$ is a transition relation: given states (q_i, p_i) and (q_j, p_j), there is a transition from (q_i, p_i) to (q_j, p_j) and we

write $(q_i, p_i) \to (q_j, p_j)$, if $q_i \xrightarrow{p_i} q_j$, i.e., there exists a transition from q_i to q_j in the mode p_i.

We can solve a switching synthesis problem for a specification given by a temporal logic formula φ in the following procedure:

1) Negate the formula φ to get $\neg\varphi$.
2) Given the transition system $(\mathcal{Q}\times\mathcal{P}, \mathcal{Q}_0\times\mathcal{P}_0, \to)$ and the LTL formula $\neg\varphi$, determine if all executions of the transition system satisfy $\neg\varphi$.

The second step above is a model checking problem and can be solved by off-the-shelf software, e.g., the SPIN model checker [17] and the NuSMV symbolic model checker [16], with computational complexity that is linear in the size of the state space [6]. Solving this problem, there are two possible outcomes: (i) the model checker verifies that $\neg\varphi$ is true for the transition system $\mathcal{T}$; (ii) the model checker finds that $\neg\varphi$ is not true and provides a counterexample.

We are particularly interested in case (ii), since it provides a switching strategy that realizes φ and therefore solves our switching synthesis problem. Actually, a counterexample given by the model checker provides either a finite or infinite path of the form

$$(q_0, p_0) \to (q_1, p_1) \to (q_2, p_2) \to (q_3, p_3) \to \cdots \tag{5}$$

that violates the formula $\neg\varphi$, or in other words, satisfies φ. A switching strategy can be extracted from a counterexample found by model checking and given in the form (5).

Switching Strategy: Given a counterexample in the form (5),

(i) if the path in (5) is infinite, we apply the switching sequence $p_0p_1p_2p_3\cdots$ to ensure that the execution

$$(q, p) = (q_0, p_0)(q_1, p_1)(q_2, p_2)(q_3, p_3)\cdots$$

satisfies φ;

(ii) if the path in (5) is finite and terminates at state (q_t, p_t), we apply any switching sequence with prefix $p_0p_1p_2p_3\cdots p_t$ to ensure that the execution

$$(q, p) = (q_0, p_0)(q_1, p_1)(q_2, p_2)(q_3, p_3)\cdots(q_t, p_t)\cdots$$

satisfies φ.

The switching protocol given by model checking is essentially an open loop strategy. It gives a mode sequence, by executing which the system is guaranteed to satisfy the specification. The correctness of the above switching strategy relies on the assumption that executions under the switching strategy can replicate the same state sequence as provided by the counterexample in the form (5). This assumption is implied if the family of transition systems (3) are an under-approximation of (1) in the sense of Definition 2. Formally, this is summarized in the following theorem.

Theorem 1. Given an under-approximation of (1), a switching strategy extracted from a counterexample found by model checking and given in the form (5) solves the discrete switching synthesis problem. This strategy can be continuously implemented to give a solution to the continuous switching synthesis problem, provided that $\left\{\mathcal{T}_p : p \in \mathcal{P}\right\}$ is an under-approximation of (1).

Proof: By definition, the switching strategy given by model checking with an under-approximation of (1) is a definite sequence of discrete states

$$(q, p) = (q_0, p_0)(q_1, p_1)(q_2, p_2)(q_3, p_3)\cdots$$

such that $(q, p) \vDash \varphi$. In view of Proposition 1, the theorem is proved, if we show that the switching strategy generates trajectories that continuously implement the discrete execution (q, p). The semantics of the above switching strategy applied to the switched system (2) are as follows. Given any initial state $x(0) \in T^{-1}(q_0)$, we let $\sigma(0) = p_0$. Therefore, the corresponding trajectory $x(t)$ follows the dynamics of $\dot{x} = f_{p_0}(x, d)$. By the definition of an under-approximation, the transition $(q_0, p_0) \xrightarrow{p_0} (q_1, p_1)$ being valid means that $\exists \tau > 0$ such that

$$x(\tau) \in T^{-1}(q_1) \quad x(t) \in T^{-1}(q_0) \cup T^{-1}(q_1), \quad t \in [0, \tau].$$

We then let $\sigma(\tau) = p_1$ and repeat the same argument. It is clear that the trajectory $x(t)$ generated this way implements (q, p) by Definition 6. ■

We remark that even if, based on an under-approximation, a model checker verifies that $\neg\varphi$ is true for the transition system $\mathcal{T}$, it does not necessarily mean that the switching synthesis problem does not have a solution. It could be the case that transition systems in (3), which serve as an abstract model for the underlying physical systems, are

too crude for the switching synthesis problem to have a solution. A finer approximation may be needed for the discrete synthesis problem to be solvable.

B. Switching synthesis by game solving

In this subsection, we consider the case in which the family of transition subsystems in (3) are an over-approximation of (1). Our approach leverages recent work on reactive synthesis [7], [26] of controllers for systems interacting with adversarial environments [27], [23], where a control protocol is synthesized to generate a sequence of control signals to ensure that a plant meets its specification for all allowable behaviors of the environment. The synthesis problem is viewed as a two-player game between the environment and the plant: the environment attempts to falsify the specification and the plant tries to satisfy it.

We propose a temporal logic game approach to switching synthesis with an abstraction that gives an over-approximation. Due to nondeterminism inherent in an over-approximation, we may not be able to exactly reason about the discrete state transitions within each mode. Rather, we seek to construct mode sequences that can force the system to satisfy a given specification despite the nondeterminism of the state transitions in each mode. A game is constructed by regarding the discrete plant variable q as the *environment part*, which tries to falsify the specification, and a switching mode p as the *controllable part*, which tries to satisfy the specification. While automatic synthesis of digital designs from general LTL specifications is one of the most challenging problems in computer science [26], for specifications in the form of the so-called Generalized Reactivity 1, or simply GR(1), formulas, it has been shown that checking its realizability and synthesizing the corresponding automaton can be accomplished in polynomial time in the number of states of the reactive system [7], [26].

We consider GR(1) specifications of the form

$$\varphi = (\varphi_q \rightarrow \varphi_s), \tag{6}$$

where, roughly speaking, φ_q characterizes the non-deterministic transitions each subsystems can make, and φ_s describes the correct behavior of the overall switching system. Here, the non-deterministic transitions of the plant, specified in φ_q, are regarded as

adversaries that try to falsify φ_s, while the switching mode is the controlled variable that tries to force the overall system to satisfy φ_s. We emphasize that, within the same framework, we can incorporate real environment into the system, by adding environment variables e that explicitly accounts for adversaries. Such adversaries do not impact the continuous dynamics of the system directly, but rather constrain its behavior through GR(1) specifications of the form

$$\varphi = ((\varphi_q \wedge \varphi_e) \rightarrow \varphi_s), \tag{7}$$

where φ_e specifies allowable environment behaviors and φ_s is a system level specification that enforces correct behaviors for all valid environment behaviors. To be more precise, for $\alpha \in \{q, s, e\}$, each φ_α in (7) has the following structure:

$$\varphi_\alpha := \varphi^\alpha_{\text{init}} \wedge \bigwedge_{i \in I^\alpha_1} \Box \varphi^\alpha_{1,i} \wedge \bigwedge_{i \in I^\alpha_2} \Box\Diamond \varphi^\alpha_{2,i},$$

where $\varphi^\alpha_{\text{init}}$ is a propositional formula characterizing the initial conditions; $\varphi^\alpha_{1,i}$ are transition relations characterizing safe, allowable moves and propositional formulas characterizing invariants; and $\varphi^\alpha_{2,i}$ are propositional formulas characterizing states that should be attained infinitely often. Many interesting temporal specifications can be transformed into this form. The readers can refer to [7], [26] for more precise treatment on how to use GR(1) game to solve LTL synthesis in many interesting cases (see also [23] for more examples). A winning strategy for the system, i.e., a strategy such that formula (7) is satisfied, can be solved by a symbolic algorithm within time complexity that is cubic in the size of the state space [7], [26].

We can formally describe our game approach for switching synthesis as follows.

Two-Player Game: A state of the game $s = (e, q, p)$ is in $\mathcal{E} \times \mathcal{Q} \times \mathcal{P}$, where $\mathcal{E}$, $\mathcal{Q}$, and $\mathcal{P}$ represent finite sets of environment states, plant states, and switching modes, respectively. A transition of the game is a move of the environment and a move of the plant, followed by a move of the switching mode. A switching strategy can be defined as a partial function $(s_0 s_1 \cdots s_{t-1}, (q_t, e_t)) \mapsto p_t$, which chooses a switching mode based on the state sequence so far and the current moves of the environment and the plant. In this sense, a switching strategy is a winning strategy for the switching system such that the specification φ is met for all behaviors of the environment and the plant. We say that

φ is *realizable* if such a winning strategy exists. If the specification is realizable, solving the two-player game gives a finite automaton that effectively gives a state-feedback switching protocol. More specifically, at each state, the system executes a switching mode, which drives the system to a number of possible states that are allowed in an over-approximation. By observing which state the system enters, the next switching mode is chosen accordingly by reading the finite automaton. Figure 2 shows a typical automaton given by solving a two-player game and how it is interpreted as a switching strategy for the system.

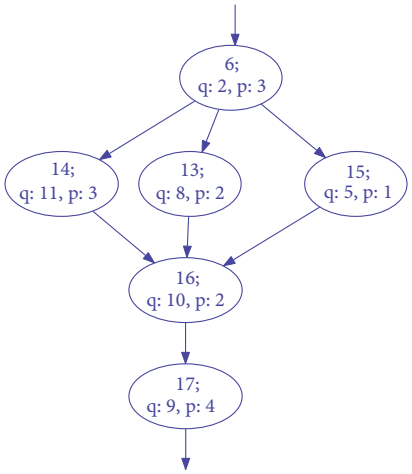

Fig. 2: Part of a synthesized automaton that can be used to extract a switching strategy. Shown in this figure, suppose that the system arrives at state number 6, where $q = 2$. It chooses mode $p = 3$ according to the automaton. Following this mode, the system may end up at three different states $q = 5$, $q = 8$, and $q = 11$, numbered as 15, 13, 14, respectively, due to non-deterministic state transitions in an over-approximation. By observing which state the system enters next, a switching mode is chosen accordingly, which eventually steers the system to a state with $q = 10$. Then mode $p = 2$ is chosen and the system enters $q = 9$, where $p = 4$ is chosen as the next mode.

By exploiting properties of an over-approximation, we can show the following result.

Theorem 2. Given an over-approximation of (1), a switching strategy obtained by solving a two-player game solves the discrete synthesis problem. This strategy can be

continuously implemented to give a solution to the continuous switching synthesis problem, provided that $\left\{\mathcal{T}_p : p \in \mathcal{P}\right\}$ is an over-approximation of (1).

Proof: By definition, the switching strategy given by solving a two-player game is a finite automaton that represents the winning switching strategy for the system. At each discrete step, the automaton provides a switching mode for all behaviors of the environment and the plant. Again, in view of Proposition 1, the theorem is proved, if we show that the switching strategy generates trajectories that continuously implement some execution (e, q, p) of the winning automata. The semantics of the above switching strategy applied to the switched system (2) are as follows. Given any initial state $x(0) \in T^{-1}(q_0)$, we choose $\sigma(0) = p_0$ from the allowable set of initial modes given by the automaton. Therefore, the corresponding trajectory $x(t)$ follows the dynamics of $\dot{x} = f_{p_0}(x, d)$. By the definition of an over-approximation, there are two possibilities. First, $x(t)$ could stay in $T^{-1}(q_0)$ for all $t \geq 0$. In this case, we must have the self-transition $q_0 \xrightarrow{p_0} q_0$ in the over-approximation. Second, there may exist some $\tau > 0$ and $q_1 \in Q$ such that

$$x(\tau) \in T^{-1}(q_1) \quad x(t) \in T^{-1}(q_0) \cup T^{-1}(q_1), \quad t \in [0, \tau].$$

In this case, we must have the transition $q_0 \xrightarrow{p_0} q_1$. In either case, we can choose the next mode from the automata, by observing the evolution of the trajectory (among the two possibilities above) and the environment's move for e. Once a next mode is generated at time τ, we repeat the same procedure from $t = \tau$. It is clear that the trajectory $x(t)$ generated this way implements one discrete execution of the winning automata. ■

Remark 3. Given a two-player game structure and a GR(1) specification, the digital design synthesis tool implemented in JTLV [28] (a framework for developing temporal verification algorithm [7], [26]) generates a finite automaton that represents a switching strategy for the system. The Temporal Logic Planning (TuLiP) Toolbox, a collection of Python-based code for automatic synthesis of correct-by-construction embedded control software as discussed in [23], [29] provides an interface to JTLV, which has been used for other applications [27], [23], [30], [29], [31], [32] and is also used to solve the examples later in this paper.

Remark 4. We have proposed two procedures for synthesizing switching protocols from temporal logic specifications: one based on model checking and the other based on temporal logic games. Model checking, of course, can be seen as a special case of game solving. Yet there are certain trade-offs among computational complexity, conservatism in models and approximations, and expressivity of specifications, which may make one approach preferable to the other. On the one hand, model checking is amenable to highly-optimized software [16], [17], with computational complexity that is linear in the size of the state space [6], but it requires an under-approximation that needs to be deterministically implemented for all allowable exogenous disturbances. Such approximations are potentially difficult to obtain. On the other hand, over-approximations account for mismatch between the continuous model and its approximation as adversarial uncertainty and model it nondeterministically. Such approximations are potentially easier to establish and also allow us to further incorporate environmental adversaries, yet the resulting formulation is a two-player temporal logic game. While games with complete LTL specifications is known to have prohibitively high computational complexity [19], we can trade computational complexity with the expressivity of specifications. Here, we focus on the GR(1) fragment of LTL. The resulting games can be solved with computational complexity that is cubic in the size of the state space [7] (with publicly available solvers [7], [18], [28] that are less evolved compared to the currently available model checkers). Further reduction in computational complexity can be obtained by restricting specifications to more limited fragments of LTL, e.g., one of the LTL formulas $\Box\psi$, $\Diamond\psi$, $\Box\Diamond\psi$, $\Diamond\Box\psi$ [34], or a boolean combination of formulas of the form $\Box\psi$ [33], where ψ is a propositional formula (see [33] for more discussions).

C. Discussions on non-Zeno implementations

We discuss in this subsection the possibility of Zeno behavior in the continuous implementations of a discrete strategy and propose appropriate assumptions to exclude such behavior. For both approaches, model checking and two-player game solving, the continuous implementations of the switching strategy correspond to an execution of the discrete strategy, which is an infinite sequence of discrete states (q, p). Particularly, for an execution of a switching strategy given by model checking, this sequence is

deterministic in the sense that it is uniquely determined by the initial state (q_0, p_0). Due to the finiteness of the abstraction, this infinite sequence consists of a possibly two parts, a finite part and a periodic part. Explicitly, we can write it as

$$(q,p) = (q_0,p_0)(q_1,p_1)\cdots(q_f,p_f)\Big((q_{f+1},p_{f+1})\cdots(q_{f+l},p_{f+l})\Big)^{\omega}, \tag{8}$$

where $l \in \mathbb{Z}^+$ and ω indicates a loop of states that are periodically repeated. According to Definition 6, a Zeno implementation of (q,p) in (8) requires that the time sequence t_k approaches some finite value as $k \to \infty$. Here each t_k corresponds to a triggering time of new discrete even (q_k, p_k) in the above infinite sequence. Therefore, Zeno behavior can happen only if the infinite loop part of the execution or a cycle can be implemented in an infinitesimal time by a continuous trajectory. The following proposition can be used to rule out Zeno implementation of the execution (q,p) in (8).

Proposition 2. If

$$\bigcap_{i=1}^{l} \overline{T^{-1}(q_{f+i})} = \emptyset, \tag{9}$$

then all continuous implementations of (q,p) in (8) are non-Zeno.

Since $T^{-1}(q) \cap T^{-1}(q') = \emptyset$ for $q \neq q'$, the above emptiness criterion is essentially on the boundaries of the cells $T^{-1}(q_{f+i})$. Furthermore, if we can check that trajectories implementing the discrete transitions can only exit within a subset of the boundaries, such as the so-called exit set in [35], the above emptiness criterion for non-Zenoness can be checked for subsets of the partitions $T^{-1}(q_{f+i})$, which give more relaxed assumptions than (9). We also remark that even if the assumption is not satisfied for all possible executions of the form (8), Zenoness can still be avoided at the discrete level by recomputing abstractions such that the assumption holds, or by adding appropriate specifications to rule out executions whose periodic part can violate (9). Particularly, since the discrete states in the periodic part typically correspond to liveness specifications, by appropriately designing abstraction and specifying liveness properties, Zenoness can be avoided.

An execution of a switching strategy given by solving a two-player game is slightly different, which is read from a finite-state automaton with states of the form (e,q,p). In this case, we look at simple cycles in the finite automaton, i.e., directed paths in the

finite automata that start and end with the same state, without no other repeated states in between. A similar criterion as (9) can be imposed to avoid Zeno executions, which assumes that $\bigcap_{i=1}^{l} \overline{T^{-1}(q_i)} = \emptyset$, where $\{(e_i, q_i, p_i) : i = 1, \cdots, l\}$ enumerates the states in a simple cycle of length l, for all simple cycles in the synthesized finite automaton.

IV. Examples

A. Temperature control

Consider a thermostat system [9], [36] with four modes, ON, OFF, Heating, Cooling, as shown in Figure 3, where the heating and cooling modes are included to capture what happens while the heater is heating up to a desired temperature and cooling down to an allowable temperature, respectively. The dynamics of the four modes are also shown in Figure 3, where x denotes the room temperature and y the temperature of the heater. In the OFF mode, the temperature changes at a rate proportional to the difference between the room temperature x and the outside temperature, which is equal to 16 in this case, according to Newton's law of cooling. In other modes, the change is at a rate proportional to the difference between the room temperature x and the temperature y of the heater. In the ON mode, the temperature of the heater is kept constant. In the heating mode, the temperature of the heater increases to 22 at a rate 0.1 per second; in the heating mode, the temperature of the heater decreases to 20 at a rate 0.1 per second.

We want the system to satisfy the following specifications:

(P1) Starting from any room temperature x and heater temperature y, the system has to reach a room temperature between 18 and 20 and a heat temperature between 20 and 22, i.e.,

$$\Diamond(18 \leq x \leq 20 \wedge 20 \leq y \leq 22).$$

(P2) If the initial temperature is already between 18 and 20, the following safety requirement is enforced

$$(18 \leq x \leq 20 \wedge 20 \leq y \leq 22) \rightarrow \Box(18 \leq x \leq 20 \wedge 20 \leq y \leq 22)).$$

(P3) Transitions among the different modes have to be in the order shown in Figure 3.

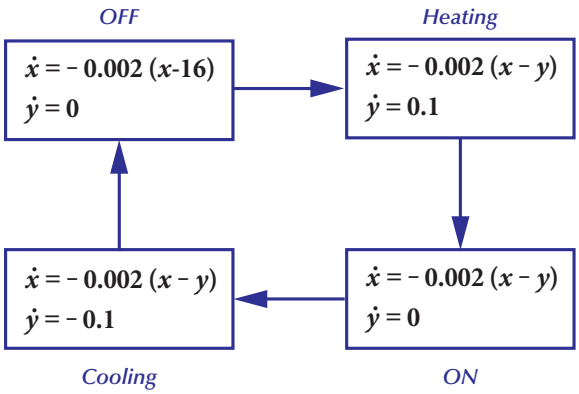

Fig. 3: A four-mode thermostat system.

The specification consists of a reachability property and an invariance property in the (x, y)-plane, together with a sequential constraint in the modes. We start with the synthesis of a switching strategy that guarantees the reachability property. To obtain a proposition preserving abstraction, we partition the plane into 12 regions as shown in Figure 4. The abstraction consists of plant variable q, whose states belong to $Q = \{q_1, \cdots, q_{12}\}$, and a mode variable p, which takes values in $\mathcal{P} = \{N, F, H, C\}$, which represent the ON, OFF, Heating, and Cooling modes, respectively. By determining the transition relations among the regions in each mode, we obtain an over-approximation of the system in the sense of Definition 3.

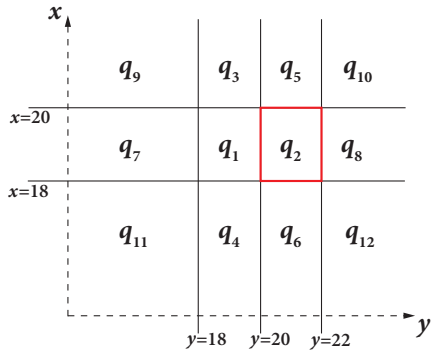

Fig. 4: A partition of the (x, y)-plane for the thermostat system for synthesizing a switching protocol that guarantees that the system reaches the region q_2.

To synthesize a switching protocol that realizes the reachability $\diamond(q_2)$, we solve a two-player game as introduced in Section III-B. The switching protocol can be extracted from a finite automaton with 32 state.

We then consider the synthesis of a switching protocol that guarantees the invariance property, i.e., $q_2 \rightarrow \Box q_2$. For this purpose, we further partition q_2 into six subregions as shown in Figure 5. We again obtain an over-approximation and solve a two-player game. The winning protocol can be extracted from a finite automaton with 7 state. A simulation result illustrating a continuous implementation of this protocol is shown in Figure 6.

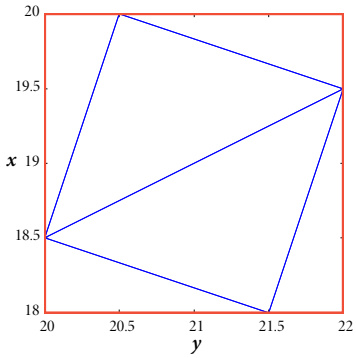

Fig. 5: A further partition of the region q_2 (indicated by the red square) in Figure 4 for synthesizing a switching protocol that achieves invariance within the region q_2.

B. Automatic transmission

Consider a 3-gear automatic transmission system [9], [36] shown in Figure 7. The longitudinal position of the car and its velocity are denoted by θ and ω, respectively. The transmission model has three different gears. For simplicity, the throttle position, denoted by u, takes value 1 in accelerating mode and -1 in decelerating modes. The specification concerns the efficiency of the automatic transmission and we use the functions

$$\eta_i(\omega) = 0.99 \exp\left(-\frac{(\omega - a_i)^2}{64}\right) + 0.01$$

to model the efficiency of gears $i = 1, 2, 3$, where $a_1 = 10$, $a_2 = 20$, $a_3 = 30$, as similarly considered in [9]. The acceleration in mode i is given by the product of the throttle

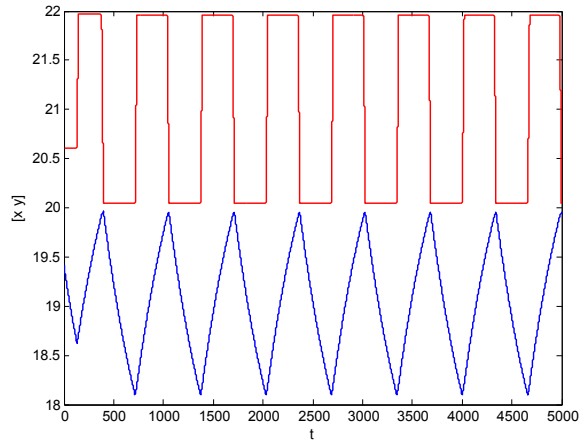

Fig. 6: A simulation illustrating the continuous implementation of a switching protocol that guarantees the invariance property $\Box(18 \leq x \leq 20 \wedge 20 \leq y \leq 22)$: the red line represents the temperature of the heater and blue line indicates the room temperature.

and transmission efficiency. The switching synthesis problem is to find a gear switching strategy to maintain a certain level of transmission efficiency when the speed is above certain value. Formally, we consider a specification

$$\Box(\omega \geq 5 \rightarrow \eta \geq 0.5) \wedge (0 \leq \omega \leq 40),$$

which consists of a minimum efficiency of 50% when the speed is greater than 5, and a speed limit of 40.

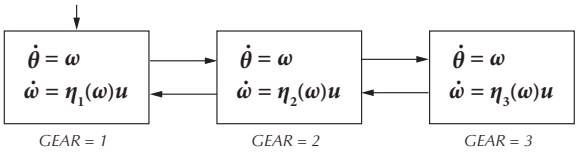

Fig. 7: A 3-gear automatic transmission.

Since the properties of interest here are related to ω only, we partition the ω-axis into a union of intervals Q that preserves the proposition on efficiency. The abstraction consists of a plant variable q, which takes values in Q, and a mode variable p, which takes values in $\mathcal{P} = \{0, \pm 1, \pm 2, \pm 3\}$. Here, $\pm$ denotes accelerating modes and decelerating modes,

respectively. We also consider a starting mode 0. According to this abstraction, we have a deterministic transition system for each of the 7 modes. Therefore, the switching synthesis problem can be solved by model checking. A simulation result is shown in Figure 8, illustrating a continuous implementation of this strategy.

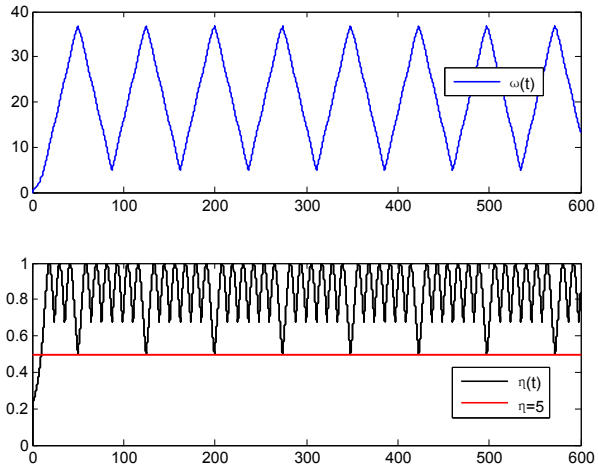

Fig. 8: Simulation results for Example 2: the upper figure shows the speed vs. time, while the lower figure shows the real-time efficiency of the transmission, where the specified level 50% is indicated by the red line.

C. *Robot motion planning*

Consider a kinematic model of a unicycle-type wheeled mobile robot [21] in 2D plane:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}. \tag{10}$$

Here, x, y are the coordinates of the middle point between the driving wheels; θ is the heading angle of the vehicle relative to the x-axis of the coordinate system; v and w are the control inputs, which are the linear and angular velocity, respectively.

To cast the motion planning of this robot as a switching synthesis problem, we consider a situation where the heading angles are restricted to a finite set $\{\theta_p : p = 1, \cdots, 8\}$, where $\theta_p \in I_p$ and I_p are non-overlapping subintervals of $[0, 2\pi)$. Here we allow the

heading angle to be within certain intervals to capture possible measurements errors or disturbances. The set of angles considered in this example is shown in Figure 9, where θ_i can be an arbitrary angle in $((i-1)\pi/4, i\pi/4)$, for $i = 1, \cdots, 8$.

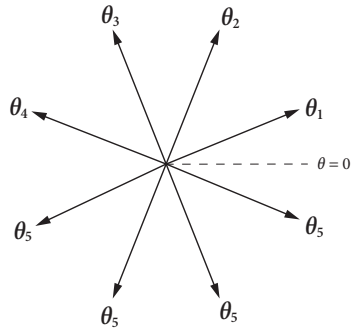

Fig. 9: Eight different heading angles of the robot.

Equation (10) can now be viewed as a switched system with four different modes

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} v_0 \cos\theta_p \\ v_0 \sin\theta_p \end{bmatrix}, \tag{11}$$

where $v_0 > 0$ is some constant speed. These dynamics can be achieved with inputs $(v, w) = (v_0, 0)$ in (10) with a desired heading angle in $\theta_p \in I_p$. Transitions between different heading angles are now regarded as mode transitions, and the transition can be rendered through $\dot{x} = \dot{y} = 0$ and $\dot{\theta} = \omega_0$, by letting inputs $(v, w) = (0, w_0)$ in (10). In this sense, transitions can be made freely among different modes.

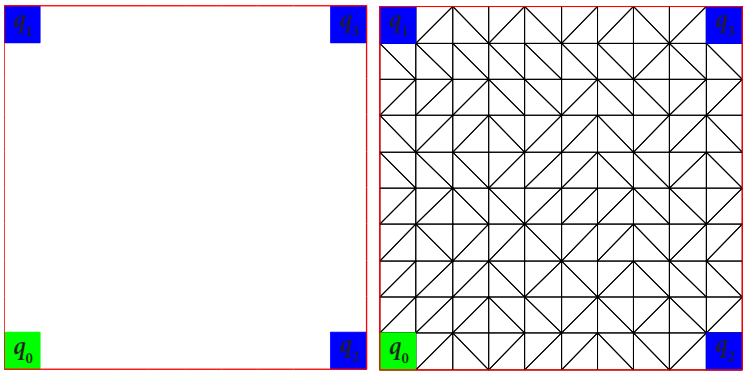

Fig. 10: The workspace for Example 3 and its partition.

We consider a workspace shown on the left side of Figure 10, which is a square of size 10. The robot is expected to satisfy the following desired properties:

(P1) Visit each of the blue cells, labeled as q_1, q_2, and q_3, infinitely often.

(P2) Eventually go to the green cell q_0 after a PARK signal is received.

Here, the PARK signal is an environment variable that constrains the behavior of the robot. The following assumption is made on the PARK signal.

(S1) Infinitely often, PARK signal is not received.

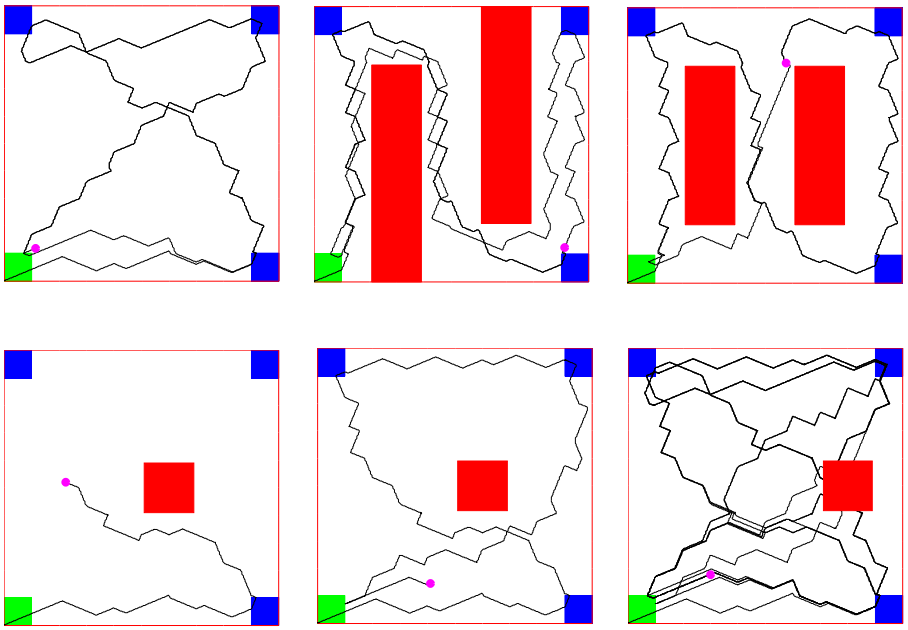

Fig. 11: Simulation results for Example 3: (a) The upper left figure shows simulation results without obstacles; (b) the upper middle and right figures show simulation results with different static obstacles; (c) the lower figures show simulation results with a moving obstacle that occupies a square of size 2 and rambles horizontally under certain assumptions on its speed. The blue squares are the regions that the robot has to visit infinitely often. The green square is where the robot should eventually visit once a PARK signal is received. The obstacles are indicated by red, the trajectories of the robot are depicted by black curves, and the current positions of the robot are represented by the magenta dots.

To synthesize a planner for this example, we introduce a partition of the workspace as shown on the right side of Figure 10, in which each cell of size 1 is partitioned into two triangles. In each mode, we can determine the discrete transition relations according to Definition 3 and obtain an over-approximation of the system. Solving a two-player game as introduced in Section III-B gives a winning strategy that guarantees that the robot satisfies the given properties (P1) and (P2). In addition, we synthesize switching strategies for a workspace occupied with both static and moving obstacles. Snapshots of simulation results are shown in Figure 11, which illustrate continuous implementations of different switching strategies that are synthesized to achieve the specification under different situations with or without obstacles.

V. Conclusions

In this paper, we considered the problem of synthesizing switching protocols for nonlinear hybrid systems subject to exogenous disturbances. These protocols guarantee that the trajectories of the system satisfy certain high-level specifications expressed in linear temporal logic. We employed a hierarchical approach where the switching synthesis problem was lifted to discrete domain through finite-state abstractions. Two different types of finite-state transition systems, namely under-approximations and over-approximations, that abstract the behavior of the underlying continuous dynamical system were introduced. It was shown that the discrete synthesis problem for an under-approximation led to a model checking problem. On the other hand, the discrete synthesis problem for an over-approximation was recast as a two-player temporal logic game. In both cases, off-the-shelf software can be used to solve the resulting problems. Moreover, existence of solutions to the discrete synthesis problem guarantees the existence of continuous implementations that are correct by construction.

As discussed in the paper, there are certain trade-offs between the fidelity of the abstractions, expressiveness of the specifications that can be handled and computational complexity. To alleviate the latter issue and to further improve the scalability of the approach, future research directions include combining the results of this paper with a receding horizon framework and/or a distributed synthesis approach.

Appendix A
Computing Finite-State Approximations

In this section, we briefly discuss how to compute an over- and under-approximation of the family of nonlinear systems (1). In general, the procedure of computing a finite-state approximation starts with a partition of the state space. Most commonly, this partition can be done through a triangulation of the state domain X. Efficient algorithms exist for computing a triangulation for polytope-type domains [37]. We shall restrict ourselves to the case where we are given such a triangulation.

Let $\mathcal{S} = \{S_i : i = 1, \cdots, M\}$ denote a triangulation of X, i.e., $\cup_{i=1}^{N} S_i = X$. Using the notation of Definitions 2 and 3, we let $Q = \{q_i : i = 1, \cdots, M\}$ and define the abstraction map $T : X \to Q$ by $T(x) = q_i$ if and only if $x \in S_i$, for all $i = 1, \cdots, M$. In other words, $T^{-1}(q_i) = S_i$ for all i. We need to determine the transition relations $q_i \xrightarrow{p} q_j$ to obtain either an over- or under-approximation of (1).

The following definitions are adapted from [35], [38].

Definition 7. A facet F of a simplex in $\mathcal{S}$ is *blocked* in mode p if

$$\overrightarrow{n} \cdot f_p(x, d) \leq 0, \quad \forall x \in F, \quad \forall d \in D, \tag{12}$$

where $\overrightarrow{n}$ is the outward unit normal vector of F and D is the set of exogenous disturbances.

Definition 8. A trajectory x of the pth mode of (1), starting in a simplex S, *exits* S at some time $T \geq 0$, if there exists $\varepsilon > 0$ such that $x(t) \in S$, for all $t \in [0, T]$, and $x(t) \notin S$, for all $t \in (T, T + \varepsilon)$.

It is easy to see that if a trajectory exits S in the pth mode at some time T, then $X(T)$ belongs to a facet F of S that is not blocked in mode p. Moreover, x will immediately enter a simplex S' that shares the common facet F with S. We apply the following algorithm to determine the transition relations $q_i \xrightarrow{p} q_j$, for all $p \in \mathcal{P}$ and all $q_i, q_j \in Q$.

Algorithm 1. Let $\mathcal{S} = \{S_i : i = 1, \cdots, M\}$.
For $i = 1 : M$

- For all facet F of S_i and each mode $p \in \mathcal{P}$, determine whether F is blocked in mode p by evaluating $isBlocked(F, S_i, p)$.
  - If $isBlocked(F, S_i, p) = 0$, add a transition $q \xrightarrow{p} q'$, where $T^{-1}(q) = S_i$, $T^{-1}(q') = S_i'$, and S_i' is the adjacent simplex that shares the facet F with S_i.
- For each mode p, determine whether all the trajectories of the pth mode of (1) exit S by evaluating $exit(S, p)$.
  - If $exit(S_i, p) = 0$, add a self-transition $q \xrightarrow{p} q$, where $T^{-1}(q) = S_i$.

Proposition 3. Let $\left\{\mathcal{T}_p := (Q, Q_0, \xrightarrow{p}) : p \in \mathcal{P}\right\}$ be the family of finite transition systems returned by Algorithm 1. If it is deterministic in the sense that for each $q \in Q$ and $p \in \mathcal{P}$, there exists a unique $q' \in Q$ such that $q \xrightarrow{p} q'$, then it is an under-approximation of (1). Otherwise, it gives an over-approximation of (1).

Apparently, the success of the above computation relies on computationally efficient algorithms for evaluating $isBlocked(F, S_i, p)$ and $exit(S_i, p)$. In general, these can be challenging issues and are certainly beyond the scope of the current paper. For systems with affine dynamics, e.g., $f_p(x, d) = Ax + Ed + a$, reachability results on simplices developed by Habets and van Schuppen [35] can be adapted and applied to provide algorithms for computing $isBlocked(F, S_i, p)$ and $exit(S_i, p)$. Essentially, due to the affinity of the dynamics, these boil down to checking a number of linear constraints at a finite number of vertices of S_i and can be solved efficiently using linear programming. For nonlinear systems, Girard and Martin [38] recently proposed a method that extends the techniques from Habets and van Schuppen [35] by approximating a nonlinear control system by its linear interpolation at the vertices of a simplex. To apply results from [38], by assuming that the dynamics are only affected by additive disturbances, we may approximate $f_p(x, d)$ on a given simplex S by $A_p x + a_p + w$, where w is a combining effects of approximation errors and exogenous disturbances, which belongs to a bounded polytope-type set W_p. Then similar techniques as in [35] can be applied to compute $isBlocked(F, S_i, p)$ and $exit(S_i, p)$. We conclude by pointing out that both [38] and [35] solve the problem of synthesizing reachability controllers for piecewise-affine hybrid systems, whereas our current paper focuses on the synthesis of switching protocol that guarantees that a nonlinear system

satisfies a high level LTL specification. Our method shares the conservativeness of their methods in that it may fail to find a switching law even if one does exist, due to the fact that our approach relies on a finite approximation of the continuous systems, which is inevitably conservative in most cases. This should be expected since even reachability for piecewise-affine hybrid systems is undecidable [39].

References

[1] J. P. Hespanha and A. S. Morse, "Switching between stabilizing controllers," *Automatica*, vol. 38, no. 11, pp. 1905 – 1917, 2002.

[2] D. Liberzon and A. Morse, "Basic problems in stability and design of switched systems," *IEEE Control Systems Magazine*, vol. 19, no. 5, pp. 59–70, 1999.

[3] E. Frazzoli, M. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.

[4] ——, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance Control and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

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

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

[7] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive (1) designs," in *Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation*, 2006, pp. 364–380.

[8] A. Church, "Logic, arithmetic and automata," in *Proceedings of the International Congress of Mathematicians*, 1962, pp. 23–35.

[9] S. Jha, S. Gulwani, S. Seshia, and A. Tiwari, "Synthesizing switching logic for safety and dwell-time requirements," in *Proceedings of the International Conference on Cyber-Physical Systems*, 2010, pp. 22–31.

[10] A. Taly and A. Tiwari, "Switching logic synthesis for reachability," in *Proceedings of the ACM International Conference on Embedded Software*, 2010, pp. 19–28.

[11] J. Cámara, A. Girard, and G. Gössler, "Synthesis of switching controllers using approximately bisimilar multiscale abstractions," in *Proceeding of the International Conference on Hybrid Systems: Computation and Control*, 2011, pp. 191–200.

[12] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli, "Effective synthesis of switching controllers for linear systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1011–1025, 2000.

[13] T. Koo, G. Pappas, and S. Sastry, "Mode switching synthesis for reachability specifications," in *Proceeding of the International Conference on Hybrid Systems: Computation and Control*. Springer, 2001, pp. 333–346.

[14] J.-W. Lee and G. E. Dullerud, "Joint synthesis of switching and feedback for linear systems in discrete time," in *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, 2011, pp. 201–210.

[15] G. Weiss and R. Alur, "Automata based interfaces for control and scheduling," in *Proceeding of the International Conference on Hybrid Systems: Computation and Control*. Springer, 2007, pp. 601–613.

[16] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: a new Symbolic Model Verifier," in *Proceedings of the International Conference on Computer Aided Verification*, 1999, pp. 495–499.

[17] G. Holzmann, *Spin Model Checker, The Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[18] B. Jobstmann and R. Bloem, "Optimizations for ltl synthesis," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, 2006, pp. 117–124.

[19] A. Pnueli and R. Rosner, "On the synthesis of an asynchronous reactive module," in *Proceedings of the International Colloquium on Automata, Languages and Programming*, 1989, pp. 652–671.

[20] D. Fisman and O. Kupferman, "Reasoning about finite-state switched systems," in *Proceedings of the International Conference on Hardware and Software: Verification and Testing*, 2011, pp. 71–86.

[21] J. Toibero, F. Roberti, and R. Carelli, "Stable contour-following control of wheeled mobile robots," *Robotica*, vol. 27, no. 01, pp. 1–12, 2009.

[22] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992, vol. 1.

[23] T. Wongpiromsarn, U. Topcu, and R. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control, submitted*, 2010.

[24] A. Pnueli, "The temporal logic of programs," in *Proceedings of the Annual Symposium on Foundations of Computer Science*, 1977, pp. 46–57.

[25] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 2000.

[26] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive (1) designs," *Journal of Computer and System Sciences, to appear*, 2011.

[27] T. Wongpiromsarn, U. Topcu, and R. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proceedings of the IEEE Conference on Decision and Control*, 2009, pp. 5997–6004.

[28] A. Pnueli, Y. Sa'ar, and L. Zuck, "JTLV: A framework for developing verification algorithms," in *Proceedings of the International Conference on Computer Aided Verification*, vol. 6174, 2010, pp. 171–174, associated tool available at `http://jtlv.ysaar.net/`.

[29] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. Murray, "TuLiP: a software toolbox for receding horizon temporal logic planning," in *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, 2011, pp. 313–314.

[30] T. Wongpiromsarn, U. Topcu, and R. Murray, "Formal synthesis of embedded control software: Application to vehicle management systems," in *Proceedings of the AIAA Infotech@Aerospace Conference*, 2011.

[31] N. Ozay, U. Topcu, T. Wongpiromsarn, and R. Murray, "Distributed synthesis of control protocols for smart camera networks," in *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems*, 2011.

[32] N. Ozay, U. Topcu, and R. Murray, "Distributed power allocation for vehicle management systems," in *Proceedings of the IEEE Conference on Decision and Control, to appear*, 2011.

[33] R. Alur and S. La Torre, "Deterministic generators and games for ltl fragments," *ACM Transactions on Computational Logic (TOCL)*, vol. 5, no. 1, pp. 1–25, 2004.

[34] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, "Controller synthesis for timed automata," in *Proceedings of the IFAC Symposium on System Structure and Control*, 1998, pp. 469–474.

[35] L. Habets, P. Collins, and J. van Schuppen, "Reachability and control synthesis for piecewise-affine hybrid systems on simplices," *IEEE Transactions on Automatic Control*, vol. 51, no. 6, pp. 938–948, 2006.

[36] J. Lygeros, "Lecture notes on hybrid systems," ETH Zurich, Tech. Rep., Dec. 2006.

[37] J. O'Rourke, *Computational Geometry in C*. Cambridge Univ Press, 1998.

[38] A. Girard and S. Martin, "Control synthesis for constrained nonlinear systems using hybridization and robust controllers on simplices," *arXiv:1103.2612*, 2011.

[39] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 94–124, 1998.