Efficient control synthesis for augmented finite transition systems with an application to switching protocols

Fei Sun, Necmiye Ozay, Eric M. Wolff, Jun Liu and Richard M. Murray

Abstract— **Augmented finite transition systems generalize nondeterministic transition systems with additional liveness conditions. We propose efficient algorithms for synthesizing control protocols for augmented finite transition systems to satisfy high-level specifications expressed in a fragment of linear temporal logic (LTL). We then use these algorithms within a framework for switching protocol synthesis for discrete-time dynamical systems, where augmented finite transition systems are used for abstracting the underlying dynamics. We introduce a notion of minimality for abstractions of certain fidelity and show that such minimal abstractions can be exactly computed for switched affine systems. Additionally, based on this framework, we present a procedure for computing digitally implementable switching protocols for continuous-time systems. The effectiveness of the proposed framework is illustrated through two examples of temperature control for buildings.**

I. Introduction

Hybrid control systems are widely used in various real-world applications, such as mode-switched flight control, vehicle management systems and automated highway systems [2]. As these applications are mostly safety-critical, the problem of correct-by-construction control synthesis for hybrid systems has been extensively studied in recent years [18]. Among these control synthesis problems, control of switched systems is of particular interest due to its practical importance [17], [10], [6], [5], [11]. The objective of switching protocol synthesis is to identify a mode sequence that allows the system to switch among a set of controllers to guarantee the satisfaction of certain specifications.

A common hierarchical approach to this problem is, first, to abstract the behaviors of continuous system as a finite transition system and, second, based on this abstraction, to solve a discrete synthesis problem to satisfy a given specification. Such approaches vary in the types of abstractions computed as well as in the nature of the discrete synthesis problem being solved. In this paper we focus on a particular type of finite transition systems, namely the augmented finite transition systems [14], that not only preserve propositional properties of the continuous systems, but are also augmented with additional liveness conditions to enforce the progress properties that are derived from the underlying continuous dynamics and essential for achieving certain specifications including reachability.

We adopt linear temporal logic (LTL) as the specification language. LTL enables us to specify a wide variety of system behaviors, such as safety, reachability, progress, stability, response and combinations of these. However, constrained by the prohibitively high computational complexity of control synthesis for full LTL specifications, we restrict the specification to a computationally efficient fragment of LTL proposed in [19]. The first contribution of this paper is to extend the algorithms developed for non-deterministic finite transition systems in [19] to efficiently synthesize control protocols for augmented finite transition systems.

In the second part of the paper, we present a framework to synthesize switching protocols for discrete-time switched systems. Within this framework, switched systems are abstracted as augmented finite transition systems; and proposed efficient algorithms are used for solving the discrete synthesis problem. We introduce a notion of minimality for the abstraction that is an indicator of how well the augmented finite transition system encodes the transitions and transience properties of underlying dynamics. We also present an extension of this framework to synthesize digitally implementable control protocols for continuous-time switched systems.

We apply the proposed techniques to two examples related to temperature control. In particular, we consider a radiant system for office buildings described by a three-dimensional switched system with thousands of discrete states in the augmented finite transition system. Based on the efficient fragment of LTL, our algorithms are able to solve discrete synthesis problems of such size, whereas a commonly used symbolic implementation tool, JTLV [16], cannot.

II. Preliminaries

In this section, we introduce the notation used throughout this paper, augmented finite transition system as the system model and fragments of linear temporal logic (LTL) as the specification language.

A. Notation

The nonnegative integers are denoted by $\mathbb{N}$ and the real numbers by $\mathbb{R}$. The n dimensional Euclidean space is denoted by $\mathbb{R}^n$. A *polytope* $\mathcal{P}$ is a convex set in $\mathbb{R}^n$, defined as $\mathcal{P} = \{x \in \mathbb{R}^n \mid Lx \leq b, \text{where}\, L \in \mathbb{R}^{m\times n},\, b \in \mathbb{R}^m\}$. For two sets X_1, X_2, their difference is denoted by $X_1 \backslash X_2$.

F. Sun is with the Information Security Center, Beijing University of Posts and Telecommunications, China. email: feisun1987@hotmail.com

N. Ozay is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA. email: necmiye@umich.edu

E. M. Wolff and R. M. Murray are with the Department of Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125, USA. emails: ewolff, murray@caltech.edu

J. Liu is with the Department of Automatic Control and Systems Engineering, University of Sheffield, Mappin Street, Sheffield S1 3JD, United Kingdom. email: j.liu@sheffield.ac.uk

For a finite set Q, Q^* denotes the set of finite sequences of elements in Q.

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), $\neg$ (not), and $\rightarrow$ (implication).

B. Augmented finite transition systems

In this paper, we consider augmented finite transition systems [14] that generalize non-deterministic transition systems by including liveness properties that enforce progress. This type of transition systems are particularly suitable for abstracting dynamical systems in that it is possible to encode transience properties of the underlying dynamics using the additional liveness properties.

Definition 1: An *augmented finite transition system* (AFTS) is a tuple $\mathcal{T} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q, \mathcal{G})$ where Q is a finite set of states, $\mathcal{A}$ is a finite set of actions (control inputs), $\rightarrow_{\mathcal{T}} \subseteq Q \times \mathcal{A} \times Q$ is a transition relation, Π is a finite set of atomic propositions, $h_Q : Q \rightarrow 2^{\Pi}$ is a labeling function and $\mathcal{G} : \mathcal{A} \rightarrow 2^{2^Q}$ a progress group map. The progress group map $\mathcal{G}$ maps each action $a \in \mathcal{A}$ to a set of subsets of Q such that the system cannot remain indefinitely within the set of states $G \in \mathcal{G}(a)$ by using only the action a. A set $G \in \mathcal{G}(a)$ is called a *progress group under action a*.

For each action $a \in \mathcal{A}$ of an AFTS $\mathcal{T}$, there is an associated directed graph $T_a = (Q, \xrightarrow{a})$, where the set $\xrightarrow{a}$ of edges corresponds to the transitions in $\rightarrow_{\mathcal{T}}$ with action a.

Definition 2: An AFTS is said to be *well-formed* if for all $a \in \mathcal{A}$ and for all $G \in \mathcal{G}(a)$, given a state $q_1 \in G$, there exists a path from q_1 to some state $q_2 \notin G$ in the graph T_a. Note that, otherwise, the system cannot make progress from the state q_1 to a state not in G, which contradicts the definition of a progress group.

An *execution* ρ of an AFTS $\mathcal{T}$ is an infinite sequence of pairs $\rho = (q(0), a(0))(q(1), a(1))(q(2), a(2))\ldots$, where $(q(i), a(i), q(i+1)) \in \rightarrow_{\mathcal{T}}$ for all $i \geq 0$ and ρ satisfies the progress conditions encoded by $\mathcal{G}$. The *run* produced by an execution ρ is an infinite sequence $\sigma = h_Q(q(0))h_Q(q(1))h_Q(q(2))\ldots$.

A *control strategy* for an AFTS $\mathcal{T}$ is a partial function $\mu : (Q \times \mathcal{A})^* \times Q \rightarrow 2^{\mathcal{A}}$ that maps the execution history to the possible next actions. A *μ-controlled execution* of $\mathcal{T}$, denoted $\mathcal{T}^{\mu}$, is an execution where for each step $i \geq 0$, the action $a(i)$ is chosen according to the control strategy μ. A *counter-strategy* (or *environment* strategy) for $\mathcal{T}$ is a partial function $\mu^{(e)} : (Q \times \mathcal{A})^* \rightarrow Q$ that resolves the non-determinism in the transitions based on the execution history.

C. Temporal logic

We use two computationally efficient fragments of linear temporal logic to specify properties such as safety, reachability, progress, and stability. These logics are Generalized Reactivity(1) (GR(1)) [15] and the fragment in [19]. The complexity of controller synthesis for both fragments is polynomial in the size of the system and specification, as opposed to doubly-exponential in the size of the specification for linear temporal logic. We first introduce the fragment in [19].

For a propositional formula φ, the notation $\bigcirc\varphi$ means that φ is true at the next step, $\square\varphi$ means that φ is always true, $\square\diamond\varphi$ means that φ is true infinitely often, and $\diamond\square\varphi$ means that φ is eventually always true [1].

Syntax: We consider formulas of the form

$$\varphi = \varphi_{\text{safe}} \ \wedge\ \varphi_{\text{act}} \ \wedge\ \varphi_{\text{per}} \ \wedge\ \varphi_{\text{rec}}, \tag{1}$$

where

$$\varphi_{\text{safe}} \doteq \square\psi_s, \qquad \varphi_{\text{act}} \doteq \bigwedge_{j \in I_r} \square(\psi_{r,j} \implies \bigcirc\varphi_{r,j}),$$
$$\varphi_{\text{per}} \doteq \diamond\square\psi_p, \qquad \varphi_{\text{rec}} \doteq \bigwedge_{j \in I_t} \square\diamond\psi_{t,j}.$$

Note that $\square\psi_s = \square \bigwedge_{j \in I_s} \psi_{s,j} = \bigwedge_{j \in I_s} \square\psi_{s,j}$ and $\diamond\square\psi_p = \diamond\square \bigwedge_{j \in I_p} \psi_{p,j} = \bigwedge_{j \in I_p} \diamond\square\psi_{p,j}$. In the above definitions, I_s, I_r, I_p, and I_t are finite index sets and $\psi_{i,j}$ and $\varphi_{i,j}$ are propositional formulas for any i and j.

Semantics: We use set operations between an execution ρ of $\mathcal{T} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q, \mathcal{G})$ and subsets of Q where particular propositional formulas hold to define satisfaction of a temporal logic formula [7]. We denote the set of states where propositional formula ψ holds by $[\![\psi]\!]$. An execution ρ *satisfies* the temporal logic formula φ, denoted by $\rho \models \varphi$, if and only if certain set operations hold.

Let ρ be an execution of the system $\mathcal{T}$, *Inf*(ρ) denote the set of states visited infinitely often in ρ, and *Vis*(ρ) denote the set of states visited at least once in ρ. Given propositional formulas ψ and φ, we relate satisfaction of a temporal logic formula of the form (1) with set operations as follows:

- $\rho \models \square\psi$ iff *Vis*$(\rho) \subseteq [\![\psi]\!]$,
- $\rho \models \diamond\square\psi$ iff *Inf*$(\rho) \subseteq [\![\psi]\!]$,
- $\rho \models \square\diamond\psi$ iff *Inf*$(\rho) \cap [\![\psi]\!] \neq \emptyset$,
- $\rho \models \square(\psi \implies \bigcirc\varphi)$ iff $\rho_i \notin [\![\psi]\!]$ or $\rho_{i+1} \in [\![\varphi]\!]$ for all i.

An execution ρ *satisfies* a conjunction of temporal logic formulas $\varphi = \bigwedge_{i=1}^{m} \varphi_i$ if and only if the set operations for each temporal logic formula φ_i holds.

A μ-controlled execution of $\mathcal{T}$ might include many possible executions because of the non-determinism. A μ-controlled execution of $\mathcal{T}$ *satisfies* the formula φ starting at state $q \in Q$, denoted $\mathcal{T}^{\mu}(q) \models \varphi$, if and only if $\rho \models \varphi$ for all $\rho \in \mathcal{T}^{\mu}(q)$. Given a system $\mathcal{T}$, state $q \in Q$ is *winning* (for the system over the "environment") for φ if there exists a control strategy μ such that $\mathcal{T}^{\mu}(q) \models \varphi$. Let $W \subseteq Q$ denote the set of winning states.

GR(1): We now (informally) define GR(1) by inheriting syntax and semantics from above. GR(1) formulas are given in the *assume/guarantee* form $\varphi \doteq \varphi^{(e)} \rightarrow \varphi^{(s)}$ where for each $\alpha \in \{e, s\}$, $\varphi^{(\alpha)}$ has the following structure:

$$\varphi^{(\alpha)} \doteq \varphi^{(\alpha)}_{\text{init}} \ \wedge\ \varphi^{(\alpha)}_{\text{safe}} \ \wedge\ \varphi^{(\alpha)}_{\text{act}} \ \wedge\ \varphi^{(\alpha)}_{\text{rec}}, \tag{2}$$

where $\varphi^{(\alpha)}_{\text{safe}}$, $\varphi^{(\alpha)}_{\text{act}}$, and $\varphi^{(\alpha)}_{\text{rec}}$, are as defined above and $\varphi^{(\alpha)}_{\text{init}} \doteq p^{(\alpha)}_0$ is a propositional formula characterizing the initial conditions.

III. Control synthesis for augmented finite transition systems

In this section, we state the main problem studied in this paper and present algorithms for solving this problem when the LTL specification φ is given in the form (1). Then, we establish the correctness of these algorithms.

A. Control synthesis problem

Problem 1: Given an AFTS $\mathcal{T}$, an initial condition $q_0 \in Q$, and a specification φ expressed in the form (1), synthesize a control strategy μ such that $\mathcal{T}^{\mu}(q_0) \models \varphi$.

We will use $(\mathcal{T}, \varphi)$ to denote an instance of Problem 1 when q_0 is not specified.

Given an augmented transition system $\mathcal{T}$, let $Post_{\mathcal{T},a}(q)$ denote the set of *a-successors* of a state q in $\mathcal{T}$, i.e., $Post_{\mathcal{T},a}(q) \doteq \{q' \mid (q,a,q') \in \rightarrow_{\mathcal{T}}\}$. The set of *successors* of q is $Post_{\mathcal{T}}(q) = \cup_{a \in \mathcal{A}} Post_{\mathcal{T},a}(q)$. Likewise, $Pre_{\mathcal{T}}(q)$ denotes the set of *predecessor* defined as $Pre_{\mathcal{T}}(q) \doteq \{q' \mid \exists a \in \mathcal{A}, (q',a,q) \in \rightarrow_{\mathcal{T}}\}$. Successors and predecessors for a set $Q' \subseteq Q$ of states are defined similarly.

B. Algorithms

For different fragments of LTL, the winning sets of control synthesis problems can be characterized in terms of controllable predecessor sets and forced predecessor sets [19], [3]. Our starting point is the characterization of winning sets from [19] for the LTL fragment in (1), however, we take a primal viewpoint to compute controllable predecessor sets and forced predecessor sets because dynamic programming based approaches as in [19] are less suitable for AFTSs.

For an AFTS $\mathcal{T}$ and a target set $B \subseteq Q$, the *controllable predecessor* set of B is defined as follows:

$$CPre^{\infty}_{\mathcal{T}}(B) \doteq \{q \in Q \mid \exists \mu, \forall \mu^{(e)}, \exists i \in \mathbb{N}, \text{s.t., the } i^{th} \text{ state of } \mu\text{-controlled execution starting at } q, \text{is in } B\}.$$

That is, for any state in $CPre^{\infty}_{\mathcal{T}}(B)$, there exists at least one control strategy that makes the system reach a state in B in a finite number of steps. Indeed, $CPre^{\infty}_{\mathcal{T}}(B)$ is the largest set that satisfies this property.

Similarly we have the *forced predecessor* set of a given set $B \subseteq Q$ on $\mathcal{T}$,

$$FPre^{\infty}_{\mathcal{T}}(B) \doteq \{q \in Q \mid \forall \mu, \exists \mu^{(e)}, \text{s.t., any } \mu\text{-controlled execution starting at } q, \text{visits } B\},$$

which includes all states that can be forced to visit B by a counter strategy $\mu^{(e)}$ independent of the control strategy.

We present the detailed procedures in Algorithm 1 and Algorithm 2 to compute, respectively, the controllable predecessor sets and the forced predecessor sets for AFTSs.

Algorithm 1 Computing the controllable predecessor set $CPre^{\infty}_{\mathcal{T}}(B)$

Input: AFTS $\mathcal{T} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q, \mathcal{G})$ and a set $B \subseteq Q$.
Output: Controllable predecessor set $CPre^{\infty}_{\mathcal{T}}(B)$ and strategy μ

```
 1: Set $C = \emptyset$, $C_n = B$
 2: Initialize $\mu(q) = \emptyset$ for all $q \in Q$
 3: while $C_n \neq C$ do
 4:     $C = C_n$
 5:     for $p \in Pre_{\mathcal{T}}(C) \setminus C$ do
 6:         for $a \in \mathcal{A}$ do
 7:             if $Post_{\mathcal{T},a}(p) \subseteq C$ then
 8:                 $\mu(p) = \{a\}$
 9:                 $C_n = C_n \cup \{p\}$
10:             else if $Post_{\mathcal{T},a}(p) \cap C \neq \emptyset$ then
11:                 for $G \in \mathcal{G}(a)$ do
12:                     if $p \in G$ then
13:                         $F = \{p\}$
14:                         for $i \in 0 : |G|$ do
15:                             $F = (Post_{\mathcal{T},a}(F) \cup F) \setminus C$
16:                         if $F \subseteq G$ then
17:                             $\mu(G) = \{a\}$
18:                             $C_n = C_n \cup F$
19: return $CPre^{\infty}_{\mathcal{T}}(B) = C$ and $\mu$
```

Algorithm 2 Computing the forced predecessor set $FPre^{\infty}_{\mathcal{T}}(B)$

Input: AFTS $\mathcal{T} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q, \mathcal{G})$, a set $B \subseteq Q$.
Output: Forced predecessor set $FPre^{\infty}_{\mathcal{T}}(B)$

```
Set $F = \emptyset$, $F_n = B$
while $F_n \neq F$ do
    $F = F_n$
    for $p \in Pre_{\mathcal{T}}(F) \setminus F$ do
        if $\forall a \in \mathcal{A}$,  $Post_{\mathcal{T},a}(p) \cap F \neq \emptyset$ then
            $F_n = F_n \cup \{p\}$
return $FPre^{\infty}_{\mathcal{T}}(B) = F$
```

Theorem 1: Given a well-formed AFTS $\mathcal{T}$ and a set $B \subseteq Q$, Algorithm 1 computes the controllable predecessor set $CPre^{\infty}_{\mathcal{T}}(B)$.

Proof: (Sketch) Algorithm 1 consists of two nested fixed point like operations. The set C is initialized with the set B. Then, for all predecessors q' of C, two things are checked. First, if there exists an action a that forces the a-successors of q' into C, then q' is part of the controllable predecessor set of C (lines 7-9). Second, if q' belongs to a progress group $G \in \mathcal{G}(a)$ and all a-successors of q' up to $|G|+1$ steps that do not enter C (F after lines 14-15) remain in G, F is part to the controllable predecessor set of C (lines 10-18). This last statement follows from the definition of progress group, that is the execution must eventually exit G if action a is used, and the fact that consecutive a-successors only exit G through C. Therefore, at each step all states added to C is in $CPre^{\infty}_{\mathcal{T}}(C)$. Finally, noting that for all $C \supseteq B$, if $C \subseteq CPre^{\infty}_{\mathcal{T}}(B)$, then $CPre^{\infty}_{\mathcal{T}}(C) = CPre^{\infty}_{\mathcal{T}}(B)$; it is possible to see that once the monotonically growing set C initialized at B convergences to a fixed point, $CPre^{\infty}_{\mathcal{T}}(B)$ is reached. ■

Theorem 2: Given an AFTS $\mathcal{T}$ and a set $B \subseteq Q$, Algorithm 2 computes the forced predecessor set $FPre^{\infty}_{\mathcal{T}}(B)$.

Proof: Algorithm 2 starts by initializing F to B. At each iteration the set of states that cannot avoid F under any action $a \in \mathcal{A}$ are added to F. Finally, noting that for all $F \supseteq B$, if $F \subseteq FPre^{\infty}_{\mathcal{T}}(B)$, then $FPre^{\infty}_{\mathcal{T}}(F) = FPre^{\infty}_{\mathcal{T}}(B)$; it is possible to see that once the monotonically growing set F initialized at B convergences to a fixed point, $FPre^{\infty}_{\mathcal{T}}(B)$ is reached. Since Q is finite, algorithm terminates. ■

As shown in [19], computing the winning set of a control synthesis problem with a specification φ expressed in the form of equation (1) reduces to a sequence of controllable predecessor set and forced predecessor set computations, together with basic set operations on these sets. Roughly speaking, controlled predecessor sets are used to check the satisfaction of reachability and liveness parts of φ and forced predecessor sets are used to establish invariance. Synthesis of corresponding strategies follows by refining or concatenating (with a finite memory in the case of multiple goals in recurrence formula φ_{rec}) of the strategies computed during the process.

While algorithm 2 does not directly return a control strategy, a strategy that guarantees invariance can be readily computed from $FPre^{\infty}_{\mathcal{T}}(B)$ as follows. Given an unsafe set B, the set $Q \setminus FPre^{\infty}_{\mathcal{T}}(B)$ is the largest controlled-invariant set that contains all the states for which there is a control strategy μ that guarantees that μ-controlled executions avoid the set B. In particular, such a strategy is given by $\mu(q) = \{a \in \mathcal{A} \mid Post_{\mathcal{T},a}(q) \subseteq Q \setminus FPre^{\infty}_{\mathcal{T}}(B)\}$, for all $q \in Q \setminus FPre^{\infty}_{\mathcal{T}}(B)$. It is also worth noting that progress groups do not have an effect on the forced predecessor sets as progress groups encode liveness properties whereas forced predecessor sets are related to safety [9]. Therefore, they are not used in Algorithm 2.

C. Complexity, comparisons and discussions

Let $n_g = \sum_{a \in \mathcal{A}} |\mathcal{G}(a)|$ be the number of progress groups in $\mathcal{T}$, $s_g = \max_{a \in \mathcal{A}, G \in \mathcal{G}(a)} |G|$ be the size of the largest progress group, $n_{out} = \max_{q,a} |Post_{\mathcal{T},a}(q)|$ be the maximum number of a-successors a state in $\mathcal{T}$ has, $n_{in} = \max_q |Pre_{\mathcal{T}}(q)|$ be the maximum number of predecessors a state in $\mathcal{T}$ has, then the complexity of Algorithm 1 is $O(n_g n_{in} n_{out} s_g |Q|)$ and $O(n_{in} n_{out} |Q|)$ for Algorithm 2.

For an AFTS $\mathcal{T}$, the progress group map $\mathcal{G}$ forces the executions of $\mathcal{T}$ to satisfy exactly the following LTL formula [14]:

$$\varphi_g \doteq \bigwedge_{a \in \mathcal{A}} \bigwedge_{G \in \mathcal{G}(a)} \neg \Diamond \Box ((\vee_{q \in G} q) \wedge a). \quad (3)$$

Using this characterization of progress groups, a synthesis problem $(\mathcal{T}, \varphi)$ for an AFTS $\mathcal{T}$ can be reduced to a synthesis problem $(\hat{\mathcal{T}}, \hat{\varphi})$ for a nondeterministic transition system $\hat{\mathcal{T}}$, where $\hat{\mathcal{T}} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q)$ and $\hat{\varphi} = \varphi_g \rightarrow \varphi$. When the original specification φ is in GR(1) fragment of LTL, so is $\hat{\varphi}$. Hence, standard implementations of GR(1) synthesis can be used. Note that $\hat{\varphi}$ includes n_g additional environment liveness assumptions with which the complexity of GR(1) synthesis scales linearly. This is consistent with the complexity of the proposed algorithm.

Algorithms 1 and 2 are very close in spirit to the fixed point operations proposed in [15] for GR(1) synthesis. However, instead of a symbolic computation as in [15], the computation is performed on explicit transition graphs as in [19]. In model-checking community, it has been observed [8] that for certain applications Binary Decision Diagram (BDD) based symbolic model-checkers (e.g., NuSMV [4]) are more efficient and for others explicit model-checkers (e.g., SPIN) are better; although there is no difference in asymptotic complexities. In this work, we have observed a similar trend in the context of synthesis. In particular, when the transition graph is sparse (i.e., the branching factor is low), the proposed algorithms perform significantly better than symbolic counterparts like JTLV [16]. Moreover, this is mostly the case for abstractions of dynamical systems.

Finally, compared to [19], the proposed algorithms can handle specifications in a slightly extended fragment than the one in Eq. (1) since they implicitly allow specific liveness assumptions of the form (3) and can be generalized to handle assumptions like Eq. (2) with an additional fixed point operation in Algorithm 1. The dynamic programming based approach in [19] is not readily suited for this type of assumptions because cost-to-go becomes ill-defined as the system can wait within a progress group arbitrarily long before making progress.

IV. Application to switching protocol synthesis

In this section, we present an application of the proposed synthesis method to switching protocols. Here, we briefly present the overall hierarchical approach for synthesizing a control strategy for switched systems. First, we construct an AFTS that abstracts the continuous behavior of the given switched system. Second, based on the finite abstraction we formulate a discrete synthesis problem. Finally, we synthesize a discrete switching protocol using the methods from Section III and implement it at the continuous level.

A. Discrete-time switched systems

A discrete-time switched affine system is a tuple $\mathcal{S} = (X, \mathcal{A}, \{f_a\}_{a \in \mathcal{A}})$, where the domain (i.e. state-space) $X \subseteq \mathbb{R}^n$ is a compact set, $\mathcal{A}$ is a finite set of modes, and $\{f_a\}_{a \in \mathcal{A}}$ is a family of vector fields satisfying $f_a(x) = A_a x + K_a$. The evolution of the system is governed by

$$x(k+1) = A_{a(k)} x(k) + K_{a(k)}. \quad (4)$$

A *trajectory* of $\mathcal{S}$ is a pair (x, a) where the state sequence $x : \mathbb{N} \rightarrow X$ and the mode sequence a: $\mathbb{N} \rightarrow \mathcal{A}$ satisfy Eq. (4) for all $k \in \mathbb{N}$. A switching protocol $u : (X \times \mathcal{A})^* X \rightarrow 2^{\mathcal{A}}$, is a feedback law that determines the possible values of $a(k)$ based on the state at time k and the trajectory up to time $k-1$.

B. Problem statement

We are interested in designing switching protocols for switched systems that satisfy a given LTL specification. In order to reason about the trajectories of $\mathcal{S}$ using LTL, we define a set $\Pi = \{\pi_0, \pi_1, \ldots, \pi_{n_p}, \pi_{out}\}$ of atomic propositions where each proposition corresponds to a polytope $\mathcal{P}_i \in X$. These propositions on the states represent the regions of interest in the state space X. Let $h_X : \mathbb{R}^n \rightarrow 2^{\Pi}$ denote the corresponding observation map such that $\pi_i \in h_X(\theta)$ if and only if $\theta \in \mathcal{P}_i$. For all $\theta \in \mathbb{R}^n \setminus X$, $h_X(\theta) = \{\pi_{out}\}$. A switched system decorated by propositions is denoted by

$\mathcal{S} = (X, \mathcal{A}, \{f_a\}_{a\in\mathcal{A}}, \Pi, h_X)$. Now we formally state the problem of switching protocol synthesis.

Problem 2: Given a switched system $\mathcal{S}$, an initial condition $x_0 \in X$ and a linear temporal logic specification φ of the form (1), synthesize a discrete switching protocol such that solutions (x, a) of $\mathcal{S}$ starting at x_0 satisfy φ.

An instance of this problem is denoted by $(\mathcal{S}, \varphi)$.

C. Abstraction via over-approximation

We start by partitioning the continuous state space X into a partition $P = \{\mathcal{P}_i\}_{i=1}^{N}$. The partition is said to be *proposition preserving*, if for all continuous states in $\mathcal{P}_i$, the same atomic propositions $\pi_i \in \Pi$ are true, i.e., $\theta_1, \theta_2 \in \mathcal{P}_i$, $\pi_i \in h_X(\theta_1)$ iff $\pi_i \in h_X(\theta_2)$.

Next, based on the proposition preserving partition, we construct the AFTS that abstracts the continuous behavior of the switched system $\mathcal{S}$. For switched systems, since there do not exist external control inputs, we need to know the inherent properties of dynamics before establishing the AFTSs. We call a subset of X a *transient region* if any trajectory starting from a state in this region eventually leaves this region. In the corresponding finite transition system, the discrete states mapped from the transient regions need to progress accordingly.

Next, we give definitions of finite-state approximations (abstractions) for switched systems extending the ideas in [14] to discrete-time setting.

Definition 3: An AFTS $\mathcal{T} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q, \mathcal{G})$ is said to be an *over-approximation* for the switched system $\mathcal{S} = (X, \mathcal{A}, \{f_a\}_{a\in\mathcal{A}}, \Pi, h_X)$ if there exist a function $\alpha : \mathbb{R}^n \rightarrow Q$ such that the following conditions hold.

(i) For all $\theta \in X$, $h_X(\theta) = h_Q(\alpha(\theta))$. There exists a unique $q_{out} \in Q$ such that $\pi_{out} \in h_Q(q_{out})$ and for some $\theta \in \mathbb{R}^n \backslash X$, we have $\alpha(\theta) = q_{out}$.

(ii) Given $q, q' \in Q$ and $q' \neq q$, there is a transition $(q, a, q') \in \rightarrow_{\mathcal{T}}$, if there exists $\theta_0 \in \alpha^{-1}(q)$ such that $f_a(\theta_0) \in \alpha^{-1}(q')$. For all $a \in \mathcal{A}$, $(q_{out}, a, q_{out}) \in \rightarrow_{\mathcal{T}}$.

(iii) The progress group map $\mathcal{G}$ is such that given an action $a \in \mathcal{A}$, for all $G \in \mathcal{G}(a)$, the set $\cup_{q\in G}\alpha^{-1}(q)$ is transient on mode a of $\mathcal{S}$.

In the above definition, condition (i) shows that the continuous state θ of switched system $\mathcal{S}$ and its corresponding discrete state $\alpha(\theta)$ in the finite transition system should be mapped to the same propositions which transforms the propositions of interest from continuous states into discrete states. For instance, a proposition preserving partition induces such a function α which maps each region in the partition to a discrete state. Condition (ii) means a transition (q, a, q') will be included in $\rightarrow_{\mathcal{T}}$ as long as there is a possible trajectory implemented by the subsystem f_a of $\mathcal{S}$ in one step. The transient property of the underlying dynamics is represented in condition (iii), which demonstrates for each action a every state in progress group $\mathcal{G}(a)$ should be mapped as the transient region of $\mathcal{S}$ on mode a. The notion of the progress group is important for switched systems in order to prevent the execution of $\mathcal{T}$ from repeating spurious cycles indefinitely. Note also that by Def. 3, any AFTS that is an over-approximation of a switched system is well-formed.

The following result follows from [14].

Theorem 3: Given an instance $(\mathcal{S}, \varphi)$ of Problem 2 and an over-approximation $\mathcal{T}$ of $\mathcal{S}$, if a strategy μ solves the instance $(\mathcal{T}, \varphi)$ of Problem 1 for all initial conditions in $Q_0 \subseteq Q$, then the switching protocol u defined, with slight abuse of notation, as $u(\cdot) \doteq \mu(\alpha(\cdot))$ solves $(\mathcal{S}, \varphi)$ for all initial conditions in $X_0 = \cup_{q\in Q}\alpha^{-1}(q)$.

Therefore, we can use properly constructed AFTSs to solve the switching protocol synthesis problem.

The next definition introduces a qualitative notion for over-approximations.

Definition 4: An AFTS $\mathcal{T} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q, \mathcal{G})$ is said to be a *minimal* over-approximation of a switched system $\mathcal{S}$ with respect to a proposition preserving partition P, if it is an over-approximation and for all over-approximations $\mathcal{T}' = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}'}, \Pi, h_Q, \mathcal{G}')$ of $\mathcal{S}$, we have $\rightarrow_{\mathcal{T}} \subseteq \rightarrow_{\mathcal{T}'}$; and for any action $a \in \mathcal{A}$, for all $G \in \mathcal{G}(a)$, there exists $G' \in \mathcal{G}'(a)$ such that $G \supseteq G'$.

Def. 4 first illustrates that a *minimal* over-approximation has minimal transitions in $\rightarrow_{\mathcal{T}}$ that only include the exact transitions in accordance with the dynamics. In other words, a transition will be included in $\rightarrow_{\mathcal{T}}$ if and only if there is a trajectory strictly implemented by the subsystem f_a of $\mathcal{S}$ in one step and therefore $\rightarrow_{\mathcal{T}} \subseteq \rightarrow_{\mathcal{T}'}$. The second part of the definition shows $\mathcal{G}(a)$ includes all possible subsets of states mapped from all transient regions of $\mathcal{S}$ under action $a \in \mathcal{A}$.

D. Computation of abstractions

Given a switched system $\mathcal{S}$ and a proposition preserving partition P, an over-approximation of $\mathcal{S}$ can be computed by Algorithm 3.

Algorithm 3 Abstraction Procedure

```
Input: switched system $\mathcal{S} = (X, \mathcal{A}, \{f_a\}_{a\in\mathcal{A}}, \Pi, h_X)$, a proposition preserving partition $P = \{\mathcal{P}_i\}_{i=0}^{N}$.
Output: AFTS $\mathcal{T} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q, \mathcal{G})$.
Let $\alpha$ be the mapping induced by $P$
Set $Q = \{\alpha(\mathcal{P}_0), \cdots, \alpha(\mathcal{P}_N), q_{out}\}$, $h_Q = h_X \circ \alpha^{-1}$
Initialize $\rightarrow_{\mathcal{T}} = \emptyset$
for $a \in \mathcal{A}$ do
    $\mathcal{G}(a) = find\_transients(P, a)$
    $\rightarrow_{\mathcal{T}} = \rightarrow_{\mathcal{T}} \cup \{(q_{out}, a, q_{out})\}$
    for $i \in \{0, 1, \cdots, N\}$ do
        for $j \in \{0, 1, \cdots, N\}$ do
            if $f_a(\mathcal{P}_i) \cap \mathcal{P}_j \neq \emptyset$ then
                $\rightarrow_{\mathcal{T}} = \rightarrow_{\mathcal{T}} \cup \{(\alpha(\mathcal{P}_i), a, \alpha(\mathcal{P}_j))\}$
        if $f_a(\mathcal{P}_i) \backslash X \neq \emptyset$ then
            $\rightarrow_{\mathcal{T}} = \rightarrow_{\mathcal{T}} \cup \{(\alpha(\mathcal{P}_i), a, q_{out})\}$
return $\mathcal{T} = (Q, \mathcal{A}, \rightarrow_{\mathcal{T}}, \Pi, h_Q, \mathcal{G})$
```

The procedure $find\_transients$ checks all subsets of P, and if the transience of a subset in mode a can be verified, includes it in $\mathcal{G}(a)$. This can be potentially exponentially complex, however implementing relaxed versions is possible. Moreover, for most affine systems, this procedure can be performed exactly and very efficiently as follows: (i) find the set X_c of all equilibrium points (i.e., $X_c = \{x_c \mid x_c =$

$A_a x_c + K_a\}$) of f_a, (ii) set $\mathcal{G}(a) = \{\{\alpha(\mathcal{P}_i) \mid \mathcal{P}_i \in P, \mathcal{P}_i \cap X_c = \emptyset\}\}$. Regions that satisfy $\mathcal{P}_i \cap X_c \neq \emptyset$ are called *critical regions*.

Theorem 4: For a switched affine system $\mathcal{S}$ and a proposition preserving partition P, if the system does not have any poles on the unit circle, a minimal over-approximation of $\mathcal{S}$ with respect to P can be computed via Algorithm 3 and the simple procedure described above for $find\_transients$.

Remark 1: Algorithm 3 is not restrictive to switched affine systems as in Eq. (4). It can be applied to nonlinear systems as well, if an outer-approximation of the post of a given set under the nonlinear dynamics can be computed. It is also possible to handle systems subject to uncertainties and disturbances.

Remark 2: Most of the set operations in Algorithm 3 can be easily computed when the proposition preserving partition consists of convex polytopes.

V. Digital switching protocols for continuous-time systems

In this section, we show how the proposed framework can be used to synthesize digitally implementable switching protocols for continuous-time switched systems.

A. Continuous-time switched systems

The continuous-time switched systems can be written as:

$$\dot{x}(t) = A^*_{a(t)} x(t) + K^*_{a(t)}, t \geq 0, \tag{5}$$

where $x(t) \in X \subseteq \mathbb{R}^n$ is the state, X is a compact set, $a(t) \in \mathcal{A}$ is the mode of the system at time t and $K^*_{a(t)}$ is the offset term. We can also describe the continuous-time system as the tuple $\mathcal{S} = (X, \mathcal{A}, \{f^*_a\}_{a \in \mathcal{A}}, \Pi, h_X)$ where $f^*_a(x) = A^*_a x + K^*_a$. The problem of switching protocol synthesis for the continuous-time systems is to generate a controller to ensure that starting from any state in the domain, the continuous-time trajectories of (5) satisfies a given LTL formula φ.

B. Discrete-time switched systems

We can use the exact solutions of (5) to derive the following discrete-time system

$$\hat{x}(k+1) = \hat{A}_{\hat{a}(k)} \hat{x}(k) + \hat{K}_{\hat{a}(k)}, \tag{6}$$

where $\hat{x}(k) = x(kT_s)$ and $\hat{a}(k) = a(kT_s)$ for all $k \geq 0$, $T_s > 0$ is a fixed sampling period, $\hat{A}_a = e^{A^*_a T_s}$, and $\hat{K}_a = \int_{kT_s}^{(k+1)T_s} e^{A^*_a((k+1)T_s - \tau)} d\tau K^*_a$. In other words, the values of states of (6) correspond to those of (5) at sampling times.

C. Relations between continuous-time systems and discrete-time systems

Instead of using the over approximation method to directly establish the AFTS for the continuous-time system, we use the discrete-time model discretized from the continuous-time system to synthesize the controller. Then we determine the relationship between these two systems in order to digitally implement the discrete control strategy on the continuous-time system created by its discrete-time model. This is summarized in the following result. Let $\delta = MT_s/2$, where $M = \max_{x \in X, a \in \mathcal{A}} \|f^*_a(x)\|$. We use φ_δ to denote the δ-contraction of a given LTL formula (which essentially means a stronger version of φ by a margin δ; see [12] for a precise definition).

Theorem 5: If we can find a strategy to make the solutions of the discrete-time switched system (6) satisfy φ_δ, then this strategy can be digitally implemented to ensure the solutions of the continuous-time switched system (5) satisfy φ.

The proof essentially follows from Proposition 2 in [12]. Thus, given a specification φ and a continuous-time switched system, we can use the approaches in Section III and IV to synthesize the controller from its discrete-time model with respect to certain contraction of the specification and digitally implement it to ensure the trajectories of continuous-time system satisfy φ. In other words, whenever we can find a switching strategy for realizing φ_δ on the time-discretized system, we can also use this strategy to realize φ on the continuous-time system. Note that M can be computed by solving $|\mathcal{A}|$ linear programs when the dynamics in each mode is affine and δ can be chosen as small as desired provided that T_s is small enough.

VI. Implementation details

Some details of the implementation are presented next.

A. Numerical considerations

In order to establish more reachability relations, we refined the proposition preserving partition P by adding grids that increase the number of regions in the partition without changing any property of the underlying dynamics. Then we created the AFTSs based on the refined proposition preserving partition. We started refining the partition with a relatively large grid size and check whether the specification was realizable. If not, we kept refining with a smaller grid size until the specification was realizable.

For the critical region, instead of declaring a region that contains an equilibrium point critical, it is possible to treat equilibrium points as additional regions, which are critical. However, sets of equilibrium points for affine systems are affine subspaces which are typically sets with empty interior. Thus, for numerical stability of polytopic computations, we expanded the set of equilibrium points by some $\epsilon > 0$ to create critical regions.

B. A heuristic for minimizing switching

High frequency switching is often undesirable for switched systems. We aim at generating a switching controller that uses a minimal number of switches to realize the given specification. In order to reduce the switch frequency, for the controllable predecessor set $CPre^\infty_\mathcal{T}(B)$, for each state in the winning set we compute all actions that makes progress towards the target set B in the outer loop in Algorithm 1 by keeping track of those actions relevant to progress groups. For the forced predecessor set $FPre^\infty_\mathcal{T}(B)$, the safe set is $Q \backslash FPre^\infty_\mathcal{T}(B)$ and the set of all safe actions are already included in the strategy for each state.Then when choosing the next action from a state q at step i, we use the action used

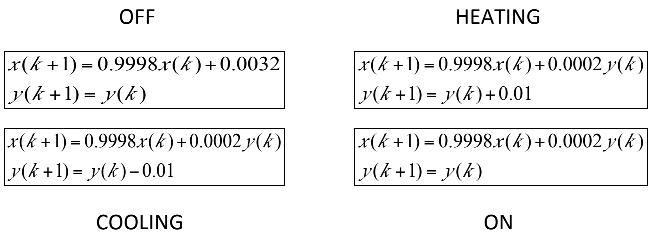
OFF
$x(k+1) = 0.9998x(k) + 0.0032$
$y(k+1) = y(k)$
HEATING
$x(k+1) = 0.9998x(k) + 0.0002y(k)$
$y(k+1) = y(k) + 0.01$
$x(k+1) = 0.9998x(k) + 0.0002y(k)$
$y(k+1) = y(k) - 0.01$
COOLING
$x(k+1) = 0.9998x(k) + 0.0002y(k)$
$y(k+1) = y(k)$
ON

Fig. 1. A four-mode thermostat system.

at step $i-1$ if this action is available instead of arbitrarily picking a possible action from $\mu(q)$.

VII. Examples

In this section, we demonstrate the proposed techniques on two examples. We used the TuLiP toolbox [20] for polytope operations and generating the controller from GR(1) formulas via the interface to JTLV [16]. The proposed synthesis algorithms for the fragment in Eq. (1) are implemented in Python. All computations were done on a MacBook Pro 2.5GHz with 8GB RAM.

A. Thermostat system

We first consider a thermostat system also used in [11] which is a continuous-time switched system. We discretize the dynamics with sampling time 0.1 seconds. The discrete-time thermostat system with four modes, ON, OFF, HEATING and COOLING, is shown in Fig. 1, where x represents the room temperature and y is the heater temperature. The domain for the state variables are taken to be $15 \leq x \leq 24$ and $15 \leq y \leq 24$.

We consider a specification for a maintenance test where the thermostat is required to change the room temperature between two ranges LOW ($17 \leq x \leq 19$ and $20 \leq y \leq 22$) and $HIGH$ ($21 \leq x \leq 22$ and $20 \leq y \leq 22$). The corresponding LTL formula is given by $\varphi_1 \doteq \Box\Diamond LOW \wedge \Box\Diamond HIGH$.

To create an AFTS, we started with a proposition preserving partition. We then added regions corresponding to equilibria with an $\epsilon = 0.1$ expansion. Note that both ON and OFF modes have non-unique equilibria for this system. We used a grid size of 0.5 to refine the partition further. Algorithm 3 was used to create the AFTS which included 416 discrete states and one progress group per mode that includes all non-critical regions within that mode. The procedure for abstraction took 609 seconds.

We synthesized controllers both with the proposed algorithm and JTLV. The former took 5 seconds and the latter took 19 seconds. A simulation result is shown in Fig. 2. Although the proposed algorithm was faster than JTLV in this example, the main advantage of the our method will be highlighted in the next example which requires a larger discrete state-space.

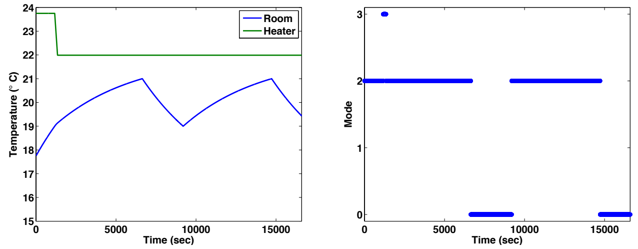

Fig. 2. Simulation results. Left: The continuous trajectories of the room temperature and the heater temperature. Right: The control inputs (i.e., the mode sequence) where we used the number $0, 1, 2, 3$ to denote the mode OFF, HEATING, ON and COOLING, respectively.

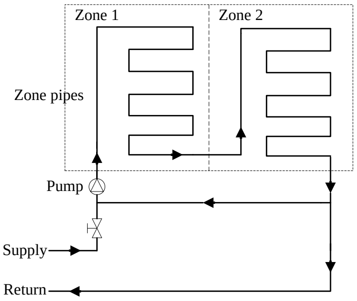

Fig. 3. Diagram of a radiant system for two zones.

B. Radiant systems in buildings

In this section, we consider a hydronic radiant system [13] for buildings. In the hydronic radiant system, hot or chilled supply water is pumped through the system tubes (i.e., the piping system) to adjust the temperature of the room.

Figure 3 shows two zones equipped with one radiant system. The dynamics of this system can be modeled as a switched system with two modes and with states T_c (temperature of the pipe), T_1, T_2 (temperatures of zones). When the pump is circulating water in its piping system, Mode 0 dynamics are active:

$$C_r\dot{T}_c = \sum_{i=1}^{2} K_{r,i}(T_i - T_c) + K_w(T_w - T_c),$$
$$C_i\dot{T}_i = K_{r,i}(T_c - T_i) + K_i(T_a - T_i) + \sum_{j\neq i} K_{ij}(T_j - T_i) + q_i,$$

for $i = 1, 2$, where T_a is the ambient air temperature and T_w is the temperature of the supply water. The other parameters are listed in Table I (see [13] for details). Similarly, when the pump is not running, Mode 1 dynamics are active, which is essentially the same as Mode 0 with K_w set to zero.

We set the domain as $20 \leq T_c, T_{1,2} \leq 27$. The specification is to get the zone temperatures to a desired range, denoted by a proposition SET ($21 \leq T_c \leq 27$ and $22 \leq T_{1,2} \leq 26$), and guarantee invariance in this range. The LTL formula for this specification is $\varphi_2 \doteq \Diamond\Box SET$.

To compute the abstraction, we started with a proposition preserving partition. This system has an equilibrium point at $(20.09, 22.75, 22.64)$ for Mode 0, and the equilibrium point for Mode 1 is outside the domain. A region is added around

TABLE I

PARAMETER VALUES FOR THE RADIANT SYSTEM.

$T_w = 18°\text{C}$	$T_a = 30°\text{C}$
$q_1 = 6$ (W/m^2)	$q_2 = 8$ (W/m^2)
$K_{12} = 1/0.2 = 5$ (W/Km^2)	
$K_1 = 1/2.1 = 0.48$ (W/Km^2)	$K_2 = 1/2.2 = 0.45$ (W/Km^2)
$K_w = 1/0.05 = 20$ (W/Km^2)	$C_r = 4000$ (kJ/K)
$K_{r,1} = 1/0.125 = 8$ (W/Km^2)	$K_{r,2} = 1/0.130 = 7.7$ (W/Km^2)
$C_1 = 1900$ (kJ/K)	$C_2 = 2100$ (kJ/K)

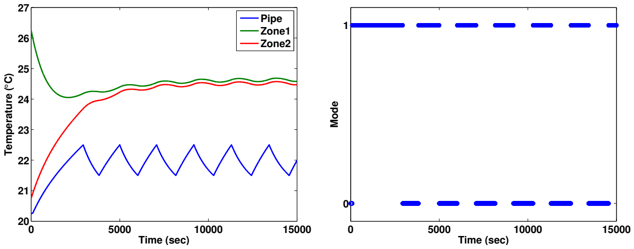

Fig. 4. Simulation results. Left: The continuous trajectories of the pipe temperature T_c and the zone temperatures $T_{1,2}$. Right: The control inputs (i.e., the mode sequence) .

the equilibrium point with $\epsilon = 0.1$ expansion. The refinement grid size was 0.5. Running Algorithm 3 resulted in an AFTS $\mathcal{T}_1$ with 2746 discrete states.

Note that persistence specifications are not in GR(1). Instead, we tried a simple GR(1) representable sufficient condition in JTLV. We also tried a recurrence formula. However, for an AFTS of this size, JTLV ran out-of-memory in all cases. On the other hand, with the proposed algorithms, synthesis took only 52 seconds. Fig. 4 depicts a simulation trajectory that starts at an initial state outside SET, eventually reaches SET and remains there as expected.

To compare quality of different abstractions, we constructed two additional AFTSs, $\mathcal{T}_2$, $\mathcal{T}_3$, on the same partition with the same transitions but smaller progress groups. All progress groups of $\mathcal{T}_2$ were set to empty sets. And, those of $\mathcal{T}_3$ were all singletons corresponding to regions without equilibrium points. Therefore, $\mathcal{T}_1$ is minimal but $\mathcal{T}_2$ and $\mathcal{T}_3$ are not. The number of winning states for the specification φ_2 for $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ were respectively 1570, 768 and 768, where the SET was satisfied in 768 states. This illustrates how the number of states that can be controlled to reach a target set increases when using minimal over-approximations.

VIII. CONCLUSIONS

In this paper we presented a framework for control strategy synthesis for AFTSs with respect to a given specification expressed in an efficient fragment of LTL. The algorithm uses an explicit graph-based representation of AFTSs and works on this graph to compute fixed points. Our experiments show that when the transition graph is sparse, the algorithm is faster and uses significantly less memory compared to BDD-based symbolic implementations. In order to apply the proposed approach to switching protocol synthesis, in the second part of the paper, we proposed an algorithm for abstracting switched systems as AFTSs. For discrete-time switched affine systems, we show that it is possible to compute a minimal AFTS that contains the exact transition relations and inherits all transience properties. We then solve the discrete synthesis problem to ensure the trajectories of the system fulfill the specification. We also considered the problem of digital implementation on continuous-time systems and based on the relations between continuous-time systems and discrete-time system, it is possible to directly construct the continuous controller from its discretized model dictated by the discrete strategy. Future work will include improving the efficiency of the abstraction process.

Acknowledgements: The authors wish to thank Prof. Yixian Yang for helpful suggestions and Dr. Vasu Raman for discussions on symbolic implementations of GR(1) synthesis. This work is supported in part by IBM and UTC through iCyPhy consortium. The work of EMW is supported by an NDSEG fellowship and the Boeing corporation.

REFERENCES

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

[2] M.S. Branicky. Analyzing and synthesizing hybrid control systems. In *Lectures on Embedded Systems*, pages 74–113. Springer, 1998.

[3] K. Chatterjee and M Henzinger. An $O(n^2)$ time algorithm for alternating Buchi games. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM*, 2012.

[4] A. Cimatti et al. NuSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364, 2002.

[5] A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Trans. on Autom. Control*, 55(1):116–126, 2010.

[6] E.A. Gol, X.C. Ding, M. Lazar, and C. Belta. Finite bisimulations for switched linear systems. In *CDC*, pages 7632–7637, 2012.

[7] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[8] G.J. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*. Addison-Wesley, 2003.

[9] O. Kupferman and M.Y. Vardi. Model checking of safety properties. In *Computer Aided Verification*, pages 172–183. Springer, 1999.

[10] D. Liberzon. *Switching in systems and control*. Birkhauser, Boston, 2003.

[11] J. Liu, N. Ozay, U. Topcu, and R.M. Murray. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Trans. on Autom. Control*, 58(7):1771–1785, 2013.

[12] J. Liu, U. Topcu, N. Ozay, and R.M. Murray. Reactive controllers for differentially flat systems with temporal logic constraints. In *IEEE CDC*, 2012.

[13] T.X. Nghiem, G.J. Pappas, and R. Mangharam. Event-based green scheduling of radiant systems in buildings. In *ACC*, 2013.

[14] N. Ozay, J. Liu, P. Prabhakar, and R.M. Murray. Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems. In *ACC*, 2013.

[15] N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive (1) designs. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 364–380, 2006.

[16] A. Pnueli, Y. Sa'ar, and L. Zuck. JTLV: A framework for developing verification algorithms. In *International Conference on Computer Aided Verification*, volume 6174, pages 171–174, 2010. Associated tool available at http://jtlv.ysaar.net/.

[17] A. Van Der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Springer-Verlag, 2000.

[18] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer-Verlag New York Inc, 2009.

[19] E.M. Wolff, U. Topcu, and R.M. Murray. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *ICRA*, 2013. revised version available at http://www.cds.caltech.edu/~ewolff/wolff_icra13_r.pdf.

[20] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R.M. Murray. TuLiP: a software toolbox for receding horizon temporal logic planning. In *HSCC*, pages 313–314, 2011.