A Bisimulation-like Algorithm for Abstracting Control Systems

Andrew J. Wagenmaker and Necmiye Ozay

Abstract— **Motivated by the recent interest in abstraction-based correct-by-construction control synthesis, in this paper we propose a new algorithm to construct finite abstractions for "large", possibly infinite, transition systems. As opposed to the standard bisimulation algorithms that create a partition of the state space, the new algorithm uses overlapping subsets of the state space as the states of the abstraction. We show that the output of the new bisimulation-like algorithm preserves realizability of linear-time properties. Several interesting properties of the algorithm are analyzed. In particular, when a finite bisimulation of the original system exists, the new algorithm is shown to always terminate in a finite number of steps. Moreover, we show with an example that even when the original system does not have a finite bisimulation, the new algorithm can result in a finite transition system whose infinite traces are equivalent to those of the original system. In the second part of the paper, we focus on the application of this algorithm to construct finite abstractions for discrete-time linear control systems and discuss several of its advantages over the standard bisimulation algorithm. Finally, the new algorithm is compared to the existing algorithms with some numerical examples.**

I. INTRODUCTION

Abstraction-based control synthesis has attracted considerable attention in the past decade as a principled means to design controllers that enforce complex specifications [1]. Roughly speaking an abstraction for a control system is a system with a fewer number of states that preserves certain properties of the concrete control system so that a controller synthesized for the abstraction can be implemented on the concrete system. For systems with continuous state spaces, this is generally accomplished by discretizing the continuous state space of the system and replacing the continuous dynamics with a finite transition system [2], [3], [4], [5]. Discrete control protocol synthesis can then be performed on these abstracted systems, producing a controller that is guaranteed to meet the specification on the continuous system. Ideally, one also wants to be able to determine the non-existence of controllers to satisfy a given specification for a concrete system just by examining the abstract system. One such relation between abstract and concrete systems that is useful to reason about existence and non-existence of controllers is the so-called bisimulation relation [1].

While algorithms already exist to compute bisimulations of arbitrary systems [6], [7], [8], finite bisimulations are only guaranteed to exist for specific classes of continuous systems [4], [7], [9], [10], [11] and, as such, these algorithms do not in general terminate in a finite number of steps. The bisimulation relation is a strong relation in the sense that it preserves both linear-time and branching-time properties. Moreover, it provides a means to reason about both existence and non-existence of controllers. When the goal is to decide whether a system can be controlled to satisfy a given linear-time property, it is possible to derive weaker relations such as simulation relations (see, for instance [12], for the use of simulation relations in verification) which can be obtained by prematurely terminating the bisimulation algorithm and redefining the transitions [13], [14].

In this paper, we propose a bisimulation-like algorithm that can generate gradually better abstractions of the concrete system. As opposed to quotienting based algorithms that use partitions of the state-space, the proposed algorithm uses overlapping subsets of the state-space as abstract states. In the case that a finite bisimulation exists, the proposed algorithm is guaranteed to terminate in a finite number of steps. The system for which the algorithm converges to holds a dual-simulation relation with the original system—it simulates and is simulated by the original system—retaining all relevant dynamic information and providing necessary and sufficient conditions for the existence of controllers upon termination. We provide examples where no finite bisimulation exists but the proposed algorithm terminates in a finite number of steps. When specialized to state-spaces that have a geometric structure such as discrete-time linear systems where propositions are defined by convex polytopes, the algorithm is also able to preserve convexity through the iterations by avoiding the set difference operation. Finally, we provide experimental evidence that, in general, when neither algorithm terminates in a finite number of steps, our algorithm is able to produce a better representation of the original system in a sense to be made clear later in the paper.

II. PRELIMINARIES

The idea of a transition system will be used extensively in this paper.

Definition 1: (Transition System) A transition system is a tuple $T = (S, \to, \Pi, \models)$ where:

- S is a set of states.
- $\to \subseteq S \times S$ is a transition relation.
- Π is a set of atomic propositions.
- $\models \subseteq S \times \Pi$ is a satisfaction relation.

We say a transition system is finite if $|S|$, $|\to|$, and $|\Pi|$ are finite and infinite otherwise. For the purposes of this paper, we restrict ourselves to $|\Pi|$ finite. Typically, an additional term, $I \subseteq S$ which denotes initial states, is included in the definition of a transition system. In this paper, we consider

This work was supported in part by NSF grants CNS-1446298 and ECCS-1553873 and the University of Michigan Summer Undergraduate Research in Engineering (SURE) program.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, `ajwagen, necmiye@umich.edu`.

every state in S a possible initial state and so eliminate I from the definition.

Associated with every transition system is a proposition preserving partition, $\mathcal{P}$, where $\bigcup_{P\in\mathcal{P}} P = S$ and for all $s_1, s_2 \in P$, $s_1 \models a \in \Pi$ implies $s_2 \models a$. If a region P preserves propositions, we overload the satisfaction relation "$\models$" and say $P \models a$. Such a proposition preserving partition is the coarsest possible proposition preserving partition if, for any $P \in \mathcal{P}$, P is the unique state satisfying some $\pi \subseteq \Pi$. In other words, $P \models a$ if and only if $a \in \pi$ and there does not exist $P' \in \mathcal{P}, P \neq P'$ such that $P' \models a$ for all $a \in \pi$. This implies, for some $s \in S$, $s \models a$ for all $a \in \pi$ if and only if $s \in P$.

Definition 2: (Trace) Given some system $T = (S, \rightarrow, \Pi, \models)$, a trace is a sequence $\pi_1\pi_2\ldots\pi_n$, $\pi_i \subseteq \Pi$, such that for $i = 1, 2, \ldots, n-1$, there exists $s_i, s_{i+1} \in S$ where $s_i \rightarrow s_{i+1}$ and $s_i \models a \in \Pi$ if and only if $a \in \pi_i$. We denote the set of traces of length N associated with our system as $Traces_N(T)$, the set of infinite traces on our system as $Traces_\infty(T)$, and the set of all traces as $Traces(T) = \bigcup_{i=1}^{\infty} Traces_i(T)$.

Two important relations exist between transition systems.

Definition 3: (Simulation Relation [8]) Given transitions systems $T_i = (S_i, \rightarrow_i, \Pi, \models_i), i = 1, 2$ defined over the same set of propositions Π, a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ is called a simulation relation if it satisfies the following:

a) For all $s_1 \in S_1$, there exists $s_2 \in S_2$ such that $(s_1, s_2) \in \mathcal{R}$.
b) For all $(s_1, s_2) \in \mathcal{R}$ where $s_1 \in S_1, s_2 \in S_2$, $s_1 \models_1 a \in \Pi$ if and only if $s_2 \models_2 a$.
c) If $(s_1, s_2) \in \mathcal{R}$ where $s_1 \in S_1, s_2 \in S_2$ and there is some $s_1' \in S_1$ such that $s_1 \rightarrow_1 s_1'$, then there exists $s_2' \in S_2$ such that $s_2 \rightarrow_2 s_2'$ and $(s_1', s_2') \in \mathcal{R}$.

If these conditions hold, we say that T_2 simulates T_1.

Definition 4: (Bisimulation Relation [8]) Given transitions systems $T_i = (S_i, \rightarrow_i, \Pi, \models_i), i = 1, 2$ defined over the same set of propositions Π, a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ is called a bisimulation relation if $\mathcal{R}$ is a simulation relation from T_2 to T_1 and $\mathcal{R}^{-1}$ is a simulation relation from T_1 to T_2.

Note that the standard simulation and bisimulation definitions weaken condition 1 in Definitions 3 and 4 to only require that initial states must be related to other states. Since we consider every state a possible initial state, we have altered this condition accordingly.

In this paper we seek to abstract transition systems into simpler representations primarily for the purposes of controller synthesis. Generally this involves grouping together similar states from the system we wish to abstract and correctly defining a transition relation on this new partition. Such abstractions must satisfy certain properties in order to be useful for control purposes.

Definition 5: (Abstraction) Given two transition systems, $T_1 = (S, \rightarrow_1, \Pi, \models_1)$ and $T_2 = (Q, \rightarrow_2, \Pi, \models_2)$, we say that T_1 is an abstraction of T_2 if it satisfies the following properties:

a) For all $s \in S, s \subseteq Q$. Further, $\bigcup_{s\in S} s = Q$.
b) If $q_1, q_2 \in s$ for some $s \in S$ and $q_1, q_2 \in Q$, then $q_1 \models_2 a \in \Pi$ implies $q_2 \models_2 a$. In other words, the elements of S are proposition preserving.
c) For some $s, s' \in S$, $s \rightarrow_1 s'$ if and only if for all $q \in s$ there exists $q' \in s'$, where $q, q' \in Q$, and $q \rightarrow_2 q'$.

Note that we do not require that for some $s, s' \in S, s \neq s'$, that $s \cap s' = \emptyset$.

Since the main motivation of this paper is finding abstractions to be used in controller synthesis, a few words on the controller synthesis setup and the type of specifications that are of interest are in order. First, we assume that all the transitions of the concrete system are *controllable*, that is, if the system is in state s and a transition exists from state s to s', then the controller can choose to take this transition. In other words, for each transition, there exists a control action to implement that transition. For the purposes of abstraction, we choose to abstract the action labels out of our representation. Second, we are primarily interested in linear-time specifications such as those given in linear temporal logic (LTL) [8]. LTL provides a rich language in which many relevant properties can be specified [13], [15], [16]. In addition, tools already exist to compute a controller to meet a LTL formula on a discrete system [3], [17]. For the purposes of this paper, it is important to note that LTL formulas are only evaluated on infinite traces [8]. This implies that two systems with the same set of infinite traces will meet the same LTL specifications. As such, when controller synthesis is being performed, we are only concerned with infinite traces and we require a valid controller to lead to only infinite traces. Deadlocking states—states with no outgoing transitions—can simply be ignored as they produce finite traces and valid controllers will avoid such states.

If a system meets the properties outlined in Definition 5 with respect to some other system, any controller synthesized to meet an LTL specification on the abstraction will also meet the same LTL specification on the original system [14]. Note that bisimulation is a much stronger relation. It preserves traces of atomic propositions and therefore also preserves LTL specifications [8]. Further, it is important to note that, given two systems T_1 and T_2, it is possible T_1 simulates T_2 and T_2 simulates T_1 but T_1 and T_2 are not bisimilar [8].

For the subsequent discussion, we introduce the following operator.

Definition 6: (Pre Operator) Given transition system $T = (S, \rightarrow, \Pi, \models)$ and some state $s \in S$, define $Pre(s) = \{s' \in S : s' \rightarrow s\}$. We overload this to handle sets $A \subseteq S$ as $Pre(A) = \{s' \in S : \exists s \in A \text{ s.t. } s' \rightarrow s\}$.

III. EXISTING BISIMULATION ALGORITHM

A well known algorithm already exists to compute a transition system with the least possible number of states that is bisimilar to a given transition system [6], [7], [8]. The algorithm operates on both finite and infinite transition systems and is guaranteed to find a finite bisimulation if one exists.

Given a transition system, the algorithm initializes with the coarsest possible proposition preserving partition of the system. It then proceeds to iterate over this partition, determining which transitions are valid and successively refining the partition at each iteration. This procedure is given in Algorithm 1.

The satisfaction relation $\models^N$ returned by Algorithm 1 can be trivially defined since the partition is proposition preserving. For some $s \in S^N, a \in \Pi$, we have $s \models^N a$ if and only if, for all $q \in s$, $q \models a$. Note that this is equivalent to stating $s \models^N a$ if and only if there exists $q \in s$ for which $q \models a$. We define the transition relation returned by Algorithm 1, $\rightarrow^N$, in accordance with Definition 5c. For some $s_1, s_2 \in S^N$, we have $s_1 \rightarrow^N s_2$ if and only if, for all $q_1 \in s_1$, there exists some $q_2 \in s_2$ such that $q_1 \rightarrow q_2$.

It can be easily shown that Algorithm 1 produces a transition system that is an abstraction of the original system as defined in Definition 5. In addition, if Algorithm 1 is prematurely terminated and a transition relation is computed using the procedure outlined above, this transition system will also be an abstraction of the original system.

As was previously mentioned, while Algorithm 1 will terminate in a finite number of steps for a finite transition system [8], it is only guaranteed to do so for special classes of infinite transition systems. For the purposes of controller synthesis, we are usually concerned with infinite transition systems. As such, when using Algorithm 1 to abstract infinite systems, Wongpiromsarn et al. [13] proposes immaturely terminating the algorithm before termination and performing controller synthesis on this intermediate system. The definition of the transition relation given in Definition 5c, when applied to a prematurely terminated system, is consistent with [13] and guarantees that the transitions of the abstract system remain controllable. Therefore, controller synthesis can be performed by model-checking as opposed to using two-player games [3], [13], [14], [18]. [1]

Ideally, if intermediate systems are to be used for controller synthesis, we would like every iteration of Algorithm 1 to produce a system that "better" approximates the original system than what was produced at the previous iteration. However, in general this is not true for Algorithm 1. It is possible to show that the system produced at the ith iteration of Algorithm 1 may contain transitions that are not realizable on the system produced at the $i + 1$th iteration. Dynamic information may be lost from iteration to iteration.

A secondary issue with Algorithm 1 is that it produces a system with a stronger relation to the original system than is necessary for control and verification purposes with linear-time properties [12]. Since our primary concern is with satisfying LTL specifications and LTL specifications are by definition only evaluated on infinite traces, in order to synthesize controllers or perform verification on a system the strongest relation we need between the abstraction and original system is for the sets of infinite traces associated with each system to be identical. As we will show, it is possible to alter Algorithm 1 such that it guarantees equivalence of infinite traces but does not require bisimilarity.

[1]Note that for the purposes of verification, we must redefine our transition relation so that, for some $s_1, s_2 \in S^i$, where S^i is our partition possibly terminated prematurely, $s_1 \rightarrow^i s_2$ if and only if there exists some $q_1 \in s_1$ and $q_2 \in s_2$ such that $q_1 \rightarrow q_2$. This transition relation allows us to verify the non-existence of unwanted traces. Most of the results in this paper are valid when we modify the transition relation in this way and one is concerned with the verification problem. In this paper we focus on the control synthesis problem, which amounts to verifying the existence of a desired infinite trace. Hence, we use the definition of transitions from [13]. Also note that when Algorithm 1 terminates in a finite number of steps, the two versions coincide on the partition produced by Algorithm 1.

Algorithm 1 Original Bisimulation Algorithm

```
1: function BISIMULATION(transition system $T = (Q, \rightarrow, \Pi, \models)$)
2:     $S^0 \leftarrow$ Coarsest possible proposition preserving partition of $T$
3:     while $\exists s_1, s_2 \in S^i$ s.t. $s_1 \cap Pre(s_2) \neq \emptyset$ and $s_1 \cap Pre(s_2) \neq s_1$ do
4:         $S^{i+1} = (S^i \backslash s_1) \cup (s_1 \cap Pre(s_2)) \cup (s_1 \backslash Pre(s_2))$
5:     end while
6:     Construct $\rightarrow^N, \models^N$
7:     return $T^N = (S^N, \rightarrow^N, \Pi, \models^N)$
8: end function
```

IV. DUAL-SIMULATION ALGORITHM

With these issues in mind, we propose a modified bisimulation-like algorithm, Algorithm 2.

Algorithm 2 Dual-Simulation Algorithm

```
1: function DUAL-SIMULATION(transition system $T = (Q, \rightarrow, \Pi, \models)$)
2:     $S^0 \leftarrow$ Coarsest possible proposition preserving partition of $T$
3:     while $\exists s_1, s_2 \in S^i$ s.t. $s_1 \cap Pre(s_2) \neq \emptyset$, $s_1 \cap Pre(s_2) \neq s_1$, and $s_1 \cap Pre(s_2) \notin S^i$ do
4:         $S^{i+1} = S^i \cup (s_1 \cap Pre(s_2))$
5:     end while
6:     Construct $\rightarrow^N, \models^N$
7:     return $T^N = (S^N, \rightarrow^N, \Pi, \models^N)$
8: end function
```

Similar to Algorithm 1, Algorithm 2 begins by partitioning the state space into the coarsest possible proposition preserving partition. It then iterates over the states, determining which transitions are valid and refining states accordingly. This algorithm differs from Algorithm 1 primarily in that it eliminates the use of the set-wise difference operator. Rather than taking the difference and intersection of sets, it simply takes the intersection and adds this to the partition. In order to maintain a partition encompassing the entire state space, it never removes regions from the partition and allows regions to overlap. This is in direct contrast to the Algorithm 1 for which all regions are disjoint.

Overlapping regions result in a hierarchical partition where regions may be contained entirely inside other regions. In fact, every region not in the original proposition preserving partition will be contained in a region from the original proposition preserving partition. From this it clearly follows that all regions will preserve propositions.

We utilize the same definitions for the returned satisfaction relation and transition relation, $\models^N$ and $\rightarrow^N$, as we used in Algorithm 1. As such, Algorithm 2 produces an abstraction of a system as defined in Definition 5. In addition, if Algorithm 2 is prematurely terminated and we define our transition relation using the same definition as was used when Algorithm 1 was prematurely terminated, the resulting transition system will also be an abstraction of the original system.

A. Formal Guarantees on Algorithm 2

We now present several formal guarantees on Algorithm 2.

In the remainder of this section, we denote the possibly infinite concrete system by $T_a = (Q, \rightarrow_a, \Pi, \models_a)$ and we let $T_b^i = (S^i, \rightarrow_b^i, \Pi, \models_b^i)$ be the result of running Algorithm 2 on T_a and prematurely terminating at the ith iteration. The transition relation $\rightarrow_b^i$ of T_b^i is created to satisfy Definition 5c. We say Algorithm 2 converges if there exists some finite N such that $S^N = \lim_{i\rightarrow\infty} S^i$. In the case of convergence, we simply denote our converged system as $T_b = (S, \rightarrow_b, \Pi, \models_b)$.

Proposition 1: If Algorithm 1 converges for system T_a, then Algorithm 2 converges for system T_a assuming the same initial proposition preserving partition is used.

Proof: Let $T_c = (S_c, \rightarrow_c, \Pi, \models_c)$ denote the system produced by running Algorithm 1 on T_a to convergence. Consider the system $T_b^i = (S^i, \rightarrow_b^i, \Pi, \models_b^i)$ produced by Algorithm 2 after i iterations where i is arbitrary and we simply require that Algorithm 2 has not converged. Let $s_1, s_2 \in S^i$ where $s_1 \cap Pre(s_2) = s_3$, $s_3 \neq s_1$, $s_3 \neq \emptyset$, and $s_3 \notin S^i$. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$, $\beta_1, \beta_2, \ldots, \beta_m \in S_c$ and assume $s_1 = \alpha_1 \cup \alpha_2 \cup \ldots \cup \alpha_n$ and $s_2 = \beta_1 \cup \beta_2 \cup \ldots \cup \beta_m$. By the definition of the Pre operator, we can see that $Pre(s_2) = Pre(\beta_1) \cup Pre(\beta_2) \cup \ldots \cup Pre(\beta_m)$. Note also, given that Algorithm 1 has converged, for any states $\alpha, \beta \in S_c$, either $\alpha \cap Pre(\beta) = \emptyset$ or $\alpha \cap Pre(\beta) = \alpha$.

By the distributive law of set algebra, we have:

$$\begin{aligned} s_1 \cap Pre(s_2) &= \left(\bigcup_{i=1}^{n} \alpha_i\right) \cap \left(\bigcup_{i=1}^{m} Pre(\beta_i)\right) \\ &= \bigcup_{i=1}^{n}\bigcup_{j=1}^{m} \alpha_i \cap Pre(\beta_j) \\ &= \alpha_{i_1} \cup \alpha_{i_2} \cup \ldots \cup \alpha_{i_k} \end{aligned} \tag{1}$$

for some $i_1, i_2, i_k \in \{1, \ldots, n\}$. Thus, taking the intersection of s_1 and $Pre(s_2)$ will not split any sets from S_c. Since we assume the original proposition preserving partitions, S^0, are the same for each algorithm, we know that for all $\alpha \in S_c$ there exists a unique $s \in S^0$ such that $\alpha \subseteq s$ and, for all $s' \in S^0$, $s' \neq s$, we have $s' \cap \alpha = \emptyset$. Thus, for all $s \in S^0$, we can write $s = \alpha_1 \cup \alpha_2 \cup \ldots \cup \alpha_n$ for some $\alpha_i \in S_c$. This, combined with (1), implies that for all $s \in S$, where S is the partition produced by Algorithm 2 on convergence, we can write $s = \beta_1 \cup \beta_2 \cup \ldots \cup \beta_m$ for some $\beta_i \in S_c$. Since S_c is finite, there is a finite number of combinations of elements from S_c. Further, since every state in S is a combination of states in S_c and we do not allow duplicate states in S, there must then be a finite number of states in S. Thus, Algorithm 2 converges.

■

Proposition 1 allows us to put an upper bound on the number of states that may be present in S. Let $g : 2^{\Pi} \rightarrow \mathbb{N}$ be a function assigning to each subset of propositions in Π the number of states in S_c uniquely satisfying satisfying that subset. In other words, if $g(\pi \subseteq 2^{\Pi}) = 1$, this implies there exists a single unique $s \in S_c$ where $s \models_c a \in \Pi$ if and only if $a \in \pi$. The maximum number of states in S is then given by $\sum_{\pi\in\Pi}(2^{g(\pi)} - 1)$. In practice, however, the number of states in S is generally much less than this.

For the purposes of proving that all LTL formulas are preserved by the abstraction generated by Algorithm 2, the following proposition is useful.

Proposition 2: A trace fragment of length n can occur on T_b if and only if that trace fragment can also occur on T_a.

Proof: Take some trace fragment $\pi_1\pi_2\ldots\pi_n$ that occurs on T_a where $\pi_i \subseteq \Pi$. This implies there exists some $q_1, q_2, \ldots, q_n \in Q$ such that $q_i \models_a c \in \Pi$ if and only if $c \in \pi_i$ and $q_i \rightarrow_a q_{i+1}$ for $i = 1, 2, \ldots, n-1$. Let s_n be some state in T_b such that $q_n \in s_n$. Let s_{n-1}^M be the unique state in T_b^0 such that $q_{n-1} \in s_{n-1}^M$. Note that $s_{n-1}^M \in T_b$. If $s_{n-1}^M \rightarrow_b s_n$, we already have that $\pi_{n-1}\pi_n$ is a realizable trace fragment on T_b. Assume this is not the case. Since $q_{n-1} \in s_{n-1}^M$ and $q_n \in s_n$, then $s_{n-1}^M \cap Pre(s_n) = s_{n-1}$ and $s_{n-1} \neq \emptyset$. By definition $s_{n-1} \rightarrow_b s_n$. Further, $s_{n-1} \in S_b$ to meet the termination condition of Algorithm 2. Thus, the trace fragment $\pi_{n-1}\pi_n$ is realizable on T_b. We can proceed in this way, now taking s_{n-1} as our starting point, all the way back to some s_1 proving that the entire trace fragment is realizable on T_b.

Now take some trace fragment $\pi_1\pi_2\ldots\pi_n$ that occurs on T_b where $\pi_i \subseteq \Pi$. This implies there exists some $s_1, s_2, \ldots, s_n \in S$ such that $s_i \models_b c \in \Pi$ if and only if $c \in \pi_i$ and $s_i \rightarrow_b s_{i+1}$ for $i = 1, 2, \ldots, n-1$. By the definition of what entails a transition on T_b, we have that for all $q_i \in s_i$, there exists some $q_{i+1} \in s_{i+1}$ such that $q_i \rightarrow_a q_{i+1}$. Thus, all transitions are replicated in T_a so the trace fragment is realizable on T_a. ■

Proposition 2 allows us to state the following corollary:

Corollary 1: Given an LTL specification φ, a controller that enforces φ on T_b exists if and only if a controller that enforces φ on T_a exists.

Proof: By assumption all the transitions of T_a are controllable and by construction all the transitions of T_b are controllable. Since Proposition 2 is true for arbitrary n, it implies that $Traces_\infty(T_b) = Traces_\infty(T_a)$. By definition, LTL formulas are only satisfiable on infinite traces. Therefore, any LTL specification that is satisfied on T_b will also be satisfied on T_a and any LTL specification satisfied on T_a will also be satisfied on T_b. This along with the fact that T_b is an abstraction of T_a implies that any controller that enforce a specification on T_b enforces the same specification on T_a and any controller that enforces a specification on T_a enforces the same specification on T_b (after the trivial mapping of transitions taken on one system to the other). ■

In addition to preserving trace fragments, we can also introduce a simulation relation between T_a and T_b.

Proposition 3: T_b simulates T_a as defined in Definition 3.

Proof: Let $h : Q \to S$ be a function sending $q \in Q$ to the smallest set $s \in S$ such that $q \in s$. Here we say some set $s \in S$ where $q \in s$ is the "smallest" if there does not exist $s' \in S$ such that $s' \subseteq s$ and $q \in s'$. Due to the hierarchical structure of the partition, it can be shown that such a set will exist. We propose the simulation relation $\mathcal{R} = \{(q, h(q)) : \forall q \in Q\}$. It can be easily shown that $\mathcal{R}$ satisfies all the conditions of a simulation relation given in Definition 3. ■

As we have already discussed, an issue with the standard bisimulation algorithm, Algorithm 1, is that it does not guarantee that at every iteration we will produce a system better than the system produced at the previous iteration. The following results show that our proposed algorithm is able to accomplish this:

Proposition 4: Every iteration of Algorithm 1 produces a system that contains all dynamics present on the system produced at the previous iteration. In other words, T_b^{i+1} simulates T_b^i.

Proof: Consider some $s_1, s_2, s_3 \in S_b^i$ where $s_1 \to_b^i s_2$ implying $s_1 \cap Pre(s_2) = s_1$. Let $s_2 \cap Pre(s_3) = s_2'$ where $s_2' \neq \emptyset$, $s_2' \neq s_2$, and $s_2' \notin S^i$. If we perform this refinement at the $i+1$th iteration, we obtain the partition $S^i \cup s_2'$. We can now add the transition $s_2' \to_b^{i+1} s_3$ and we clearly maintain the transition $s_1 \to_b^{i+1} s_2$ since neither s_1 nor s_2 has been altered.

Now consider instead refining s_1 with s_3. Let $s_1 \cap Pre(s_3) = s_1'$ where $s_1' \neq s_1$, $s_1' \neq \emptyset$, and $s_1' \notin S^i$. If we perform this operation at the $i+1$th iteration, we obtain the partition $S^i \cup s_1'$. We can now add the transition $s_1' \to_b^{i+1} s_3$ and the transition $s_1' \to_b^{i+1} s_2$ since $s_1 \cap Pre(s_2) = s_1$ and $s_1 \subseteq S$ which implies $s_1' \cap Pre(s_2) = s_1'$. Finally, we maintain the transition $s_1 \to_b^{i+1} s_2$ since neither s_1 nor s_2 has been altered.

From this it is clear that, at every iteration, no transitions are lost and some may be added. Thus, every iteration produces a system that contains all dynamics present on the previous system. If we introduce a relation $\mathcal{R}$ where $\mathcal{R}$ associates every state in S^i with its identical state in S^{i+1}, it is trivial to show that $\mathcal{R}$ is a simulation relation and T_b^{i+1} simulates T_b^i. ■

In addition to evolving in a way that successively produces a system better than the last, we would also like to have some guarantees on the intermediate system, T_b^i, that allow it to be used for controller synthesis.

Proposition 5: Any trace fragment found on T_b^i is also realizable on T_a

Proof: Take some trace fragment $\pi_1\pi_2\ldots\pi_n$ on T_b^i where $\pi_j \subseteq \Pi$ This implies there exists some $s_1, s_2, \ldots, s_n \in S_b^i$ such that $s_j \models_b^i a \in \Pi$ if and only if $a \in \pi_j$ and $s_j \to_b^i s_{j+1}$ for $j = 1, 2, \ldots, n-1$. By the definition of a transition, we know that there exists some $q_j \in s_j$, $q_{j+1} \in s_{j+1}$ such that $q_j \to_a q_{j+1}$. This implies $\pi_1\pi_2\ldots\pi_n$ can be realized on T_a. ■

Since n is arbitrary, Proposition 5 implies that the set of infinite traces realizable on T_b^i is a subset of the set of infinite traces realizable on T_a. That is to say, $Traces_\infty(T_b^i) \subseteq Traces_\infty(T_a)$. Due to this, while T_b^i may not exhibit the entire range of behaviors present on T_a, we can still perform controller synthesis on it since any LTL specification satisfiable on T_b^i will also be satisfiable on T_a.

Proposition 6: T_a simulates T_b^i as defined in Definition 3.

Proof: We propose the simulation relation $\mathcal{R} = \{(s, q) : \forall s \in S^i, \forall q \in s\}$. By construction of $\mathcal{R}$ and the definition of the transition relation, $\to_a$, on T_a, it can be easily shown that this relation satisfies Definition 3. ■

Corollary 2: T_a simulates T_b.

Proof: Follows directly from Proposition 6 since $T_b = T_b^N$. ■

From Corollary 2 and Proposition 3, we have that T_b simulates T_a and T_a simulates T_b. Thus, the system produced by Algorithm 2 bears a dual-simulation relation with the original system. This implies that any transition in T_b has a corresponding transition in T_a and any transition in T_a has a corresponding transition in T_b. Note that T_a and T_b are not necessarily bisimilar as defined by Definition 4. This primarily results from the fact that a deadlocking state may exist in T_b while no such state exists in T_a. Since we do not ever eliminate states from our partition, it is possible there exists some state $s_1 \in S$ such that for all $s_2 \in S$, $s_1 \cap Pre(s_2) \neq s_1$. This implies there are no outgoing transitions from s_1. If no such deadlocking state exists in T_a, s_1 will not be bisimilar to any states in T_a and no bisimulation relation between T_a and T_b will exist.

B. Discrete-Time Linear Systems

We now specialize our results to discrete-time linear systems of the form:

$$x(t+1) = Ax(t) + Bu(t) \qquad (2)$$

where $x(t) \in X \subseteq \mathbb{R}^n$ is the system state, $u(t) \in U \subseteq \mathbb{R}^m$ is the control signal, and $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are the system matrices.

We assume regions of interest $\{X_i\}$ in X are marked with atomic propositions from a set Π. Then, with every system of the form (2) we can associate an infinite transition system, $T = (X, \to, \Pi, \models)$, where $x' \to x$ if there exists $u \in U$ such that $x' = Ax + Bu$.

Given $T = (X, \to, \Pi, \models)$, one can apply Algorithm 2 to obtain an abstraction. In order to compute the Pre of a region where the system dynamics are given by (2), a projection operation from the state-input space to the state space needs to be performed. This computation is fairly standard when the region for which the Pre is computed and the input set U are convex polyhedra [19] (but not so for non-convex regions). Further, a convex partition allows controller synthesis to be performed much more efficiently since, in this case, only convex optimization problems must be solved when synthesizing a controller. This motivates the following result.

Proposition 7: If the original proposition preserving partition consists of only convex regions, every state in the partition produced by Algorithm 2 will be convex.

Proof: This follows from the fact that, given a system of the form (2), the Pre of a convex set will be convex and the intersection of two convex sets will also be convex [19], [20]. If we assume our initial partition is convex, this implies every region added to our partition will also be convex. ■

Note that Algorithm 1 does not preserve convexity. When the set-wise difference is taken between two regions, the region produced may be non-convex even if the original regions are convex [20]. This can lead to non-convex regions in the partition produced by Algorithm 1 even when the initial proposition preserving partition is convex, making each iteration of Algorithm 1 non-trivial.

It is also worth mentioning that, as a direct consequence of Proposition 1, Algorithm 2 converges in a finite number of steps for classes of discrete-time system that admit finite bisimulations [9]. Moreover, as shown in Section V-B.1, there are discrete-time linear systems that do not admit finite bisimulations but the proposed algorithm converges. Characterizing the finite step convergence properties of Algorithm 2 beyond the known convergence results for Algorithm 1 is the subject of ongoing research.

V. EXAMPLES

We next present several examples comparing Algorithm 2 with Algorithm 1. All computations were carried out in the Temporal Logic Planning Toolbox (`TuLiP`) [17].

A. Discrete System

We first contrast the systems produced by Algorithms 1 and 2 through a simple discrete example. Figure 1 illustrates the original transition system where the propositions associated with each state are given in $\{\}$. Figure 2a displays the transition system obtained from running Algorithm 1 on the original transition system and Figure 2b displays the transition system obtained from running Algorithm 2 on the original transition system.

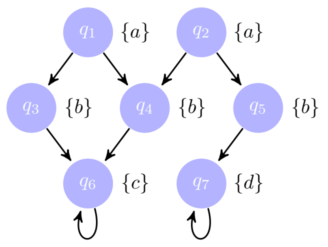
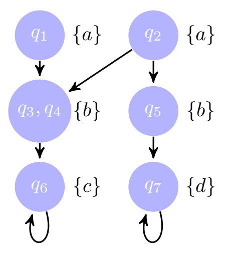
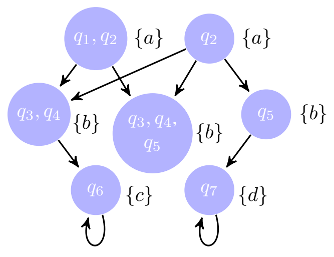


Fig. 1: Original Transition System

From Figure 2b, we can see that Algorithm 2 produces a deadlocking state when no deadlocking state exists in the original system. Due to this deadlocking state, the set of all possible traces on the original transition system and



(a) Algorithm 1



(b) Algorithm 2

Fig. 2: Systems produced by each algorithm

the system produced by Algorithm 1 differ from the set of traces on the system produced by Algorithm 2. Recall that, due to Definition 1, every state in each transition system is considered an initial state. On the original system and system produced by Algorithm 1, the set of traces is then $\{c^\omega, d^\omega, bc^\omega, bd^\omega, abc^\omega, abd^\omega\}$ where c^ω and d^ω indicate that the propositions c and d are repeated infinitely often. In contrast, the set of all possible traces on the system produce by Algorithm 2 is $\{b, c^\omega, d^\omega, ab, bc^\omega, bd^\omega, abc^\omega, abd^\omega\}$. Ignoring deadlocking states, it can be seen that the sets of infinite traces on each system are identical, guaranteeing that LTL specifications are preserved across abstractions.

B. Comparison of Convergence Properties

We have observed that, while Algorithm 2 is guaranteed to converge if Algorithm 1 converges, in many cases Algorithm 2 converges even if Algorithm 1 does not. In these cases then, Algorithm 2 is able to find a finite representation of an infinite system that preserves LTL formulas even when no finite bisimulation of such a system exists. We next present two examples illustrating this.

1) One-Dimensional System: Consider the following system:

$$x(t+1) = 2x(t) + u(t) \tag{3}$$

with state space $X = [-1.5, 1.5]$, input set $U = [-2, 2]$, and initial proposition preserving partition: $s_1 = [-1.5, -1)$, $s_2 = [-1, 1)$, $s_3 = [1, 1.5]$. We let $\Pi = \{a, b, c\}$ and assign to each region a unique proposition: $s_1 \models a$, $s_2 \models b$, $s_3 \models c$.

On this system and initial partition, Algorithm 2 converges to the following partition: $s_1 = [-1.5, -1)$, $s_2 = [-1, 1)$, $s_3 = [1, 1.5]$, $s_4 = [-1, 0.5)$, $s_5 = [-0.5, 1)$, $s_6 = [-0.5, 0.5)$, $s_7 = [-1.25, -1)$, $s_8 = [1, 1.25)$.

We now show that no finite bisimulation exists for this system—Algorithm 1 does not converge. Given a set $[a, b] \subseteq [-1.5, 1.5]$, to compute the Pre, we must find what $x \in [-1.5, 1.5]$ satisfy $2x + u \in [a, b]$. Equivalently, $a \leq 2x + u \leq b$ which implies $\frac{1}{2}(a-u) \leq x \leq \frac{1}{2}(b-u)$. Since $-2 \leq u \leq 2$, we have $\frac{1}{2}(a-2) \leq x \leq \frac{1}{2}(b+2)$. Define functions $h(x) = \frac{1}{2}(x-2)$ and $g(x) = \frac{1}{2}(x+2)$, then $Pre([a, b]) = [h(a), g(b)] \cap [-1.5, 1.5]$[2]. If we are running Algorithm 1, the operation of taking the Pre of a region and intersecting it with the existing regions can be thought of as adding points to a set containing all the boundaries. Our initial set will simply be $\{-1.5, -1, 1, 1.5\}$ and at every

[2]Half open intervals can be handled in a similar way.

iteration we will add points $h(x)$ or $g(x)$ to this set where x existed in the previous iteration of the set. Consider now the sequence:

$$a_0 = 1, \quad a_i = \begin{cases} g(a_{i-1}), & \text{if i even} \\ h(a_{i-1}), & \text{if i odd} \end{cases}$$

This can be rewritten as $a_i = \left(\frac{1}{2}\right)^i + \left(\frac{2}{3}\right)(-1)^i\left(1 - \left(\frac{-1}{2}\right)^i\right)$. Clearly, this sequence asymptotically approaches $\left(\frac{2}{3}\right)(-1)^i$ as i approaches infinity. From this we can see that our set of boundaries will never be closed under the Pre operator and therefore no finite bisimulation exists.

The fundamental difference between Algorithm 1 and Algorithm 2 is that, due to taking intersections as well as differences, Algorithm 1 allows you to apply both h and g to any boundary while Algorithm 2 does not. Consider the region $[1, 1.5]$. The Pre of this region is $[-0.5, 1.5]$ which, when intersected with the region $[-1, 1]$, produces the region $[-0.5, 1]$. Since -0.5 is only the lower boundary of a region and not the upper in the partition produced by Algorithm 2, we can only add $h(-0.5)$ to our set of boundaries and not $g(-0.5)$. In contrast, if we performed the same split using Algorithm 1, we get the two regions $[-0.5, 1]$ and $[-1, -0.5]$, allowing us to add both $h(-0.5)$ and $g(-0.5)$ to our set of boundaries. This restriction prevents us from introducing the infinite sequence we obtain by applying Algorithm 1.

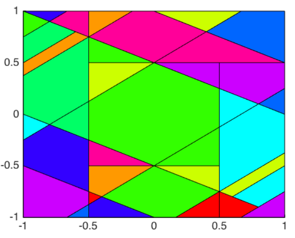
2) Two-Dimensional System: Consider now the following two-dimensional system:

$$\begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 0.5 & 1 \\ 0.75 & -1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}, \qquad (4)$$

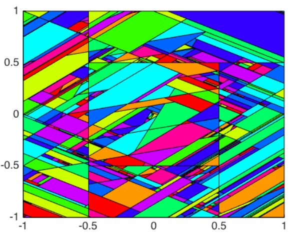
where $X = [-1, 1] \times [-1, 1]$ and $U = [-1, 1] \times [-1, 1]$. We introduce the following initial partition $s_1 = [-0.5, 0.5) \times [-0.5, 0.5)$, $s_2 = [-1, -0.5) \times [-1, 1]$, $s_3 = [-0.5, 1] \times [0.5, 1]$, $s_4 = [0.5, 1] \times [-1, 0.5)$, $s_5 = [-0.5, 0.5) \times [-1, -0.5)$. To each region we assign a unique proposition from the set $\Pi = \{a, b, c, d, e\}$. Namely: $s_1 \models a, s_2 \models b, s_3 \models c, s_4 \models d, s_5 \models e$.

When run on this system, Algorithm 2 converges to a partition containing 66 regions. Though the complexity of the dynamics prevents us from easily proving that Algorithm 1 will not converge on, when running Algorithm 1, a partition containing 700 regions was reached and the convergence criteria had still not been met. For computational and numerical reasons, the execution was terminated at this point. Regardless of whether (4) has a finite bisimulation, the intermediate partition produced by Algorithm 1 is significantly more complex than the full partition produced by 2.

The partition produced by Algorithm 2 is illustrated in Figure 3b with overlapping cells simply plotted on top of each other. The partition obtained by prematurely terminating Algorithm 1 after reaching a partition containing 700 regions is displayed in Figure 3a. From this it is clear that Algorithm 2 is able to obtain a much simpler representation of (4) than Algorithm 1 while preserving all LTL formulas.



(a) Algorithm 2



(b) Algorithm 1

Fig. 3: Partitions produced by prematuraly terminating each algorithm

C. Comparison of Intermediate Systems

Finally, we compare the performance of Algorithm 2 and Algorithm 1 when each algorithm is prematurely terminated before convergence is reached. In order to obtain an even comparison of the systems, we only allow states that have a volume greater than 0.1. This constraint forces both algorithms to ignore smaller states and artificially converge.

We consider the system:

$$\begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 0.3 & 0.9 \\ -1.1 & 0.4 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} + \begin{bmatrix} d_1(t) \\ d_2(t) \end{bmatrix} \qquad (5)$$

where $X = [-1, 1] \times [-1, 1]$ and $U = [-0.75, 0.75] \times [-0.75, 0.75]$. d_1 and d_2 represent random disturbances. We restrict the strength of the disturbances such that $|d_1| \leq d, |d_2| \leq d$ for some d.

Our initial partition is: $s_1 = [-1, 1] \times [-1, -0.1)$, $s_2 = [-1, -0.3) \times [-0.1, 0.2)$, $s_3 = [-0.3, 0.3) \times [-0.1, 0.2)$, $s_4 = [0.3, 1] \times [-0.1, 0.2)$, $s_5 = [-1, 1] \times [0.2, 1]$. We let $\Pi = \{a, b, c, d, e\}$ and assign a unique proposition to each cell: $s_1 \models a, s_2 \models b, s_3 \models c, s_4 \models d, s_5 \models e$.

We wish to somehow compare the abstractions produced by each algorithm. In the absence of a simulation relation between systems, there is no direct way to determine which transition system better represents the original system. To produce some metric of comparison between the abstractions, we propose determining the number of unique trace fragments of a given length, N, that are realizable on each system as compared to the number realizable on the original system. While this does not guarantee that more LTL formulas are satisfiable on an abstraction—since LTL only considers infinite traces—it does give some indication as to which abstraction is able to more fully capture the behaviors present on the original system. Table I illustrates these results for various values of d.

			Algorithm 2		Algorithm 1	
d	N	Possible Traces	Realizable Traces	%	Realizable Traces	%
0	5	3125	**2185**	**69.92**	773	24.74
	10	9765625	**4315343**	**44.19**	442961	4.54
0.1	5	2395	**2185**	**91.23**	316	13.19
	10	4730125	**4315343**	**91.23**	54556	1.15
0.15	5	80	**63**	**78.75**	9	11.25
	10	2560	**2048**	**80.00**	9	0.35

TABLE I: Possible Traces on Volume Constrained Abstraction

From Table I we can see that, regardless of the disturbance level, the system produced by Algorithm 2 exhibits many more possible behaviors than the system produced by Algorithm 1. In addition, Algorithm 2 is much more robust to disturbances than Algorithm 1. For Algorithm 2 the number of realizable trace fragments is unaffected when moving from no disturbance to a disturbance with strength 0.1. In contrast the number of realizable trace fragments on the abstraction produced by Algorithm 1 decreases significantly when disturbances are added. Disturbances tend to affect smaller regions more strongly. Since Algorithm 1 splits regions, it produces a partition containing smaller regions more quickly than does Algorithm 2 and is therefore more sensitive to disturbances. When $d = 0.15$, we see that the number of realizable trace fragments on each abstraction decreases yet the abstraction produced by Algorithm 2 still produces a much fuller representation of all possible trace fragments than does Algorithm 1.

Note that the number of possible traces on the concrete system decreases in the presence of disturbances since we only consider controllable transitions. For a transition to be controllable under disturbances, it must be controllable for all disturbance values in the allowable range. This restriction eliminates some transitions in the case of large disturbances that are possible to enforce when no disturbances are present.

Note also that `TuLiP` uses several randomized algorithms to handle the complexity of non-convex regions. As such, the results obtained for Algorithm 1 listed in Table I differ somewhat from trial to trial and the median values across 100 trials are given.

VI. CONCLUSION

In this paper we have proposed a novel bisimulation-like algorithm able to produce abstractions of transition systems bearing a dual-simulation relation with the original system. This algorithm utilizes an idea similar to the well known bisimulation algorithm—differing primarily in that it allows regions to overlap—but is able to improve on several of its shortcomings. Specifically, it provides guarantees on the intermediate systems produce and how these systems evolve. In addition, it is able to preserve convexity of partitions when applied to discrete-time linear systems. We have contrasted our algorithm and the standard bisimulation algorithm through several examples and have demonstrated that our algorithm may converge in situations where the standard bisimulation algorithm will not and will produce better intermediate systems when prematurely terminated.

Motivated by our experimental observations, in future work we hope to provide stronger guarantees on when our algorithm will converge for discrete-time linear systems, specifically in cases where no finite bisimulations exist. It is also of interest to investigate for what additional classes of systems this algorithm can be shown to find finite abstractions. In addition, we hope to develop a more rigorous metric that will allow us to formally compare the abstraction produced by our algorithm and other similar abstraction algorithms, especially when the corresponding abstractions are not related by a simulation relation. Finally further optimizations of the algorithm similar to those in [21] will be considered.

ACKNOWLEGMENTS

The authors would like to thank Oscar Mickelin for insightful discussions at the early stages of this work.

REFERENCES

[1] P. Tabuada, *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[2] P. Nilsson, N. Ozay, U. Topcu, and R. M. Murray, "Temporal logic control of switched affine systems with an application in fuel balancing," in *Proceedings of the 2012 American Control Conference (ACC)*, June 2012, pp. 5302–5309.

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

[4] P. Tabuada and G. J. Pappas, "Linear time logic control of discrete-time linear systems," *Automatic Control, IEEE Transactions on*, vol. 51, no. 12, pp. 1862–1877, 2006.

[5] B. Yordanov, J. Tůmová, I. Černá, J. Barnat, and C. Belta, "Temporal logic control of discrete-time piecewise affine systems," *Automatic Control, IEEE Transactions on*, vol. 57, no. 6, pp. 1491–1504, 2012.

[6] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs, "Minimal model generation," in *International Conference on Computer Aided Verification*. Springer, 1990, pp. 197–203.

[7] 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.

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

[9] P. Tabuada and G. J. Pappas, *Model checking LTL over controllable linear systems is decidable*, 2003, pp. 498–513.

[10] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. E. Dullerud, "A decidable class of planar linear hybrid systems," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2008, pp. 401–414.

[11] M. Rungger, M. Mazo Jr, and P. Tabuada, "Specification-guided controller synthesis for linear systems and safe linear-time temporal logic," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*. ACM, 2013, pp. 333–342.

[12] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke, "Computing simulations on finite and infinite graphs," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, Oct 1995, pp. 453–462.

[13] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[14] J. Liu, N. Ozay, U. Topcu, and R. M. Murray, "Switching protocol synthesis for temporal logic specifications," in *2012 American Control Conference (ACC)*, June 2012, pp. 727–734.

[15] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343 – 352, 2009.

[16] S. Jiang and R. Kumar, "Failure diagnosis of discrete-event systems with linear-time temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 49, no. 6, pp. 934–945, June 2004.

[17] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with tulip: The temporal logic planning toolbox," in *Proc. Multi-conference on Systems and Control (MSC)*, 2016, pp. 1030–1041. [Online]. Available: tulip-control.org

[18] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Synthesis of control protocols for autonomous systems," *Unmanned Systems*, vol. 1, no. 01, pp. 21–39, 2013.

[19] F. Borrelli, A. Bemporad, and M. Morari, *Model Predictive Control*, 2015.

[20] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[21] D. Bustan and O. Grumberg, "Simulation-based minimization," *ACM Transactions on Computational Logic (TOCL)*, vol. 4, no. 2, pp. 181–206, 2003.