2016 IEEE 55th Conference on Decision and Control (CDC)
ARIA Resort & Casino
December 12-14, 2016, Las Vegas, USA

Formal Synthesis of Control Strategies for Dynamical Systems

Calin Belta

Abstract— **In formal verification, the goal is to check whether the executions of a system satisfy a rich property, usually expressed as a temporal logic formula. While the formal verification problem received a lot of attention from the formal methods community for the past thirty years, the dual problem of formal synthesis, in which the goal is to synthesize or control a system from a temporal logic specification, has not received much attention until recently. This tutorial paper provides a self-contained exposition on formal synthesis of control strategies for a particular class of dynamical systems. Central to this paper is the concept of transition system, which is shown to be general enough to model a wide variety of dynamical systems. It is shown how abstractions can be constructed by using simulation and bisimulation relations. The control specifications are restricted to formulas of Linear Temporal Logic (LTL) and some fragments of LTL, which are introduced together with the corresponding automata and the automata games used to generate control strategies for finite transition systems. At the end, we show how such control strategies can be adapted to discrete-time piecewise affine systems. Several examples are provided throughout the paper.**

I. Introduction

In control theory, complex models of physical processes, such as systems of differential or difference equations, are usually checked against simple specifications, such as stability and set invariance. In formal methods, rich specifications, such as languages and formulas of temporal logics, are checked against simple models of software programs and digital circuits, such as finite transition systems. With the development and integration of cyber physical and safety critical systems, there is an increasing need for computational tools for verification and control of complex systems from rich, temporal logic specifications. For example, in a persistent surveillance application, an unmanned aerial vehicle might be required to "take photos of areas A and B infinitely often while always avoiding unsafe areas C and D." In the emergent area of synthetic biology, the goal is to design small gene networks from specifications that are naturally given as temporal logic statements about the concentrations of species of interest, *e.g.,* "if inducer u_1 is low and inducer u_2 is high, then protein y should eventually be expressed and remain in this state for all future times."

Central to the existing approaches for formal verification and control of infinite-state systems is the notion of abstraction. Roughly, an abstract model can be seen as a finite transition graph, whose states label equivalent sets of states of the original system, and whose transitions match the trajectories of the original system among the equivalence classes. Once constructed, such an abstraction can be used for verification (using off-the-shelf model checking tools) or control (using automata game techniques) in lieu of the original system.

The main objective of this tutorial paper is to present a formal synthesis algorithm for a class of discrete-time system called piecewise affine systems. Such systems are quite general, as they have been shown to approximate nonlinear system with arbitrary accuracy. There also exist computational tools for identifying such systems (both the polytopes and the corresponding dynamics) from experimental data.

This tutorial paper is based on the work of the author, and is, as a result, biased and non-comprehensive. A more comprehensive version of this, which includes verification, optimality, and other classes of dynamical systems, can be found in [1]. The specifications are restricted to formulas of Linear Temporal Logic (LTL) and fragments of LTL, even though other temporal logics have been used by other authors. While some of the results can be extended to continuous-time systems, the focus is on discrete-time systems only. We only cover deterministic and purely non-deterministic systems, even though existing results, including ours, show that extensions to stochastic systems and probabilistic temporal logics are possible. The equivalence notion that we use is classical bisimulation - extensions to approximate bisimulations and probabilistic bisimulations have been developed recently.

This tutorial paper is intended to a broad audience of scientists and engineers with interest in formal methods and controls. Computer scientists are shown that simulations and bisimulations, normally used to reduce the size of finite models of computer programs, can be used to abstract infinite-state systems. The paper also provides a self-contained exposition of temporal logic control for finite non-deterministic systems, which is useful even for seasoned formal methods researchers. Control theorists are introduced to notions such as abstractions, temporal logics, and formal synthesis, and are shown that such techniques can be used for classical systems such as discrete-time piecewise affine systems.

The paper is organized as follows. In Sec. II, we introduce (nondeterministic) transition systems, a formalism that can be used to model a large spectrum of dynamical systems. Simulation and bisimulations relations and corresponding abstractions for transitions systems are also defined in this section. The syntax and semantics of Linear Temporal Logic

The author is with the Department of Mechanical Engineering, Boston University, Boston, MA 02215 {cbelta}@bu.edu.
This work was partially supported by the NSF under grants CPS- 1446151 and CMMI-1400167.

(LTL) are introduced and illustrated with several examples in Sec. IV, together with the Büchi and Rabin automata accepting languages satisfying LTL formulas. Sec. V focuses on finite systems, *i.e.*, transition systems with finitely many states, inputs, and observations. In this section, we solve the problem of finding the largest set of states and corresponding control strategies such that all trajectories of a system satisfy an LTL formula. We show that this problem can be mapped to a Büchi game or a Rabin game, depending on the structure of the specification formula. We present ready to implement solutions to all these problems and include illustrative examples. Finally, in Sec. VI, we bring together the concepts and techniques introduced so far and present a computational framework for control of (infinite) discrete-time piecewise affine systems from LTL specifications.

II. Transition Systems

In this paper, we focus on transition systems as a modeling formalism for a wide range of processes [2]. Besides capturing the behavior of many processes directly, transition systems provide a semantical model for various high-level formalisms for concurrent systems including Kripke structures [3], process algebras [4], statecharts [5] and Petri nets [6] (also see [7] for additional discussion on discrete event systems). In addition, the formalisms of Mealy and Moore machines, which are related to finite state automata and are commonly used in hardware synthesis and analysis, can be described by transition systems [8].

In this section, we define the syntax and semantics of transition systems, and provide several illustrative examples. In particular, we present different (deterministic, nondeterministic, finite, and infinite) transition system representations for discrete-time dynamical systems. Such embeddings have been proposed by several authors [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. While such embeddings can be easily defined for continuous-time systems as well [10], [15], [16], we do not give these definitions as the focus in this tutorial paper is on discrete-time systems only.

As probabilistic dynamics are not covered in this paper, the transition systems defined here capture only purely deterministic and nondeterministic behaviors. The probabilistic version of the transition system considered here is the well known Markov decision process (MDP), in which the inputs enable transitions with given probability distributions among the states of the system [19]. Throughout this paper, we also assume that the states of the system are observable. In other words, the current state of the system is known and available for state-feedback control. Readers interested in systems for which this assumption is relaxed are referred to transition systems with nondeterministic observations [20], for which the current state is known only to belong to a given set, and probabilistic versions such as hidden Markov models (HMM) [21], partially observable Markov decision processes (POMDP) [22], and mixed observability Markov decision processes (MOMDPs) [23].

A. Definitions and Examples

Definition 1 (Transition system): A transition system is a tuple $T = (X, \Sigma, \delta, O, o)$, where

- X is a (possibly infinite) set of states,
- Σ is a (possibly infinite) set of inputs (controls or actions),
- $\delta : X \times \Sigma \to 2^X$ is a transition function,
- O is a (possibly infinite) set of observations, and
- $o : X \to O$ is an observation map.

A subset $X_r \subseteq X$ is called a *region* of T. A transition $\delta(x,\sigma) = X_r$ indicates that, while the system is in state x, it can make a transition to any state $x' \in X_r$ in region $X_r \subseteq X$ under input σ. We denote the set of inputs available at state $x \in X$ by

$$\Sigma^x = \{\sigma \in \Sigma \mid \delta(x,\sigma) \neq \emptyset\}. \quad (1)$$

A transition $\delta(x,\sigma)$ is *deterministic* if $|\delta(x,\sigma)| = 1$ and the transition system T is deterministic if for all states $x \in X$ and all inputs $\sigma \in \Sigma^x$, $\delta(x,\sigma)$ is deterministic. T is *non-blocking* if, for every state $x \in X$, $\Sigma^x \neq \emptyset$. Throughout this paper, only non-blocking transition systems are considered. Transition system T is called *finite* if its sets of states X, inputs Σ, and observations O are all finite.

An *input word* of the system is defined as an infinite sequence $w_\Sigma = w_\Sigma(1)w_\Sigma(2)w_\Sigma(3)\ldots \in \Sigma^\omega$. A *trajectory* or *run* of T produced by input word w_Σ and originating at state $x_1 \in X$ is an infinite sequence $w_X = w_X(1)w_X(2)w_X(3)\ldots$ with the property that $w_X(k) \in X$, $w_X(1) = x_1$, and $w_X(k+1) \in \delta(w_X(k), w_\Sigma(k))$, for all $k \geq 1$. We denote the set of all trajectories of a transition system T originating at x by $T(x)$. We use $T(X_r) = \cup_{x' \in X_r} T(x')$ to denote the set of all trajectories of T originating in region $X_r \subseteq X$. As a consequence, $T(X)$ will denote the set of all trajectories of T.

A run $w_X = w_X(1)w_X(2)w_X(3)\ldots$ defines an *output word* (which we will refer to simply as *word*) $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$, where $w_O(k) = o(w_X(k))$ for all $k \geq 1$. The set of all words generated by the set of all trajectories starting at $x \in X$ is called the *language* of T originating at x and is denoted by $\mathscr{L}_T(x)$. The language of T originating at a region $X_r \subseteq X$ is $\mathscr{L}_T(X_r) = \bigcup_{x' \in X_r} \mathscr{L}_T(x')$. The language of T is defined as $\mathscr{L}_T(X)$, which for simplicity is also denoted as $\mathscr{L}_T$. We often represent an infinite word as a finite *prefix* followed by an infinite *suffix* as shown in Example 1.

For an arbitrary region $X_r \subseteq X$ and set of inputs $\Sigma' \subseteq \Sigma$, we define the set of states $Post_T(X_r, \Sigma')$ that can be reached from X_r in one step by applying an input in Σ' (called *successors* of X_r under Σ') as

$$Post_T(X_r, \Sigma') = \{x \in X \mid \exists x' \in X_r,\ \exists \sigma \in \Sigma',\ x \in \delta(x', \sigma)\} \quad (2)$$

Similarly, the set of states that reach some $X_r \subseteq X$ in one step under the application of some input from $\Sigma' \subseteq \Sigma$ (called *predecessors* of X_r under Σ') can be defined as

$$Pre_T(X_r, \Sigma') = \{x \in X \mid \exists x' \in X_r,\ \exists \sigma \in \Sigma',\ x' \in \delta(x, \sigma)\} \quad (3)$$

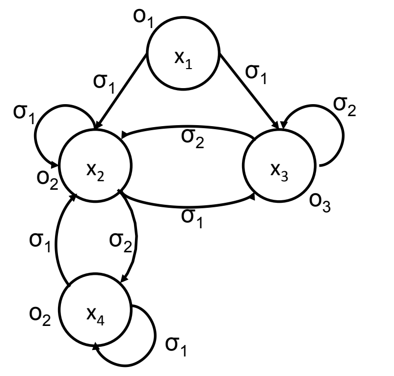


Fig. 1. Graphical representation of the transition system defined in Example 1. Each state is represented by a circle containing the state label. The observation of each state is shown close to its circle and transitions are represented by arrows between states. The input enabling a transition is shown on top of the corresponding arrow.

For a deterministic T, each state x has a single successor under a given input σ, *i.e.* $Post_T(x,\sigma)=\delta(x,\sigma)$ [1] is a singleton, but, in general, can have multiple predecessors, *i.e.*, $Pre_T(x,\sigma)$ is a region of T. However, for a nondeterministic T it is possible that both $Post_T(x,\sigma)$ and $Pre_T(x,\sigma)$ are regions of T.

Example 1: The finite, non-deterministic transition system $T=(X,\Sigma,\delta,O,o)$ shown in Figure 1 is defined formally by

- $X=\{x_1,x_2,x_3,x_4\}$,
- $\Sigma=\{\sigma_1,\sigma_2\}$,
- $\delta(x_1,\sigma_1)=\{x_2,x_3\}$, $\delta(x_2,\sigma_1)=\{x_2,x_3\}$, $\delta(x_2,\sigma_2)=\{x_4\}$, $\delta(x_3,\sigma_2)=\{x_2,x_3\}$, $\delta(x_4,\sigma_1)=\{x_2,x_4\}$,
- $O=\{o_1,o_2,o_3\}$,
- $o(x_1)=o_1$, $o(x_2)=o(x_4)=o_2$, $o(x_3)=o_3$.

The set of inputs available at the states of the system are $\Sigma^{x_1}=\{\sigma_1\}$, $\Sigma^{x_2}=\{\sigma_1,\sigma_2\}$, $\Sigma^{x_3}=\{\sigma_2\}$, and $\Sigma^{x_4}=\{\sigma_1\}$. States x_1 and x_3 form a region $X_r=\{x_1,x_3\}$. Region X_r can be reached from states x_1 and x_2 under σ_1, and, therefore, $Pre_T(X_r,\sigma_1)=\{x_1,x_2\}$. Similarly, states x_2 and x_3 are reachable from region X_r under σ_1, and therefore $Post_T(X_r,\sigma_1)=\{x_2,x_3\}$.

Starting from x_1, under input word $w_\Sigma=\sigma_1\sigma_1\sigma_2(\sigma_1)^\omega$, one possible run of the system is $w_X=x_1x_2x_2(x_4)^\omega$. Sequences $\sigma_1\sigma_1\sigma_2$, $x_1x_2x_2$ are prefixes (repeated once) and σ_1, x_4 are suffixes (repeated infinitely many times) for input word and run, respectively. This run originates in region X_r and defines an infinite (output) word $w_O=o_1(o_2)^\omega\in\mathscr{L}_T(X_r)$. There is an infinite number of infinite words in the language $\mathscr{L}_T(X_r)=\{o_1(o_2)^\omega,o_1o_2(o_3)^\omega,o_1(o_3)^\omega,(o_3)^\omega,o_3o_3(o_2)^\omega,\ldots\}$.

B. Discrete-Time Dynamical Systems as Transition Systems

Consider the following discrete-time control system:

$$\mathscr{D}:\quad \begin{array}{rcl} x_{k+1} &=& f(x_k,u_k),\ k=0,1,2,\ldots,\\ y_k &=& g(x_k), \end{array} \tag{4}$$

where $x_k\in\mathbb{R}^N$ is the state at time k, $u_k\in U\subseteq\mathbb{R}^M$ is the control input at time k (U is the control constraint set), $f:\mathbb{R}^N\times\mathbb{R}^M\to\mathbb{R}^N$ is a vector function describing the dynamics of the system, $y_k\in\mathbb{R}^P$ is the output at time k, and $g:\mathbb{R}^N\to\mathbb{R}^P$ is the output function.

The discrete-time system from Equation (4) can be easily represented as an infinite transition system $T^{1,c}_{\mathscr{D}}=(X,\Sigma,\delta,O,o)$, where

- the set of states $X=\mathbb{R}^N$,
- the set of inputs $\Sigma=U$,
- the transition function $\delta=f$ [2],
- the set of observations $O=\mathbb{R}^P$, and
- the observation map $o=g$.

The transition system $T^{1,c}_{\mathscr{D}}$ is called the one-step embedding of $\mathscr{D}$, since it captures all the transitions that $\mathscr{D}$ can take in one discrete-time step. It is also a timed and controlled embedding, because it preserves the time and control information from the original system. This is the most natural embedding of $\mathscr{D}$, since $T^{1,c}_{\mathscr{D}}$ and $\mathscr{D}$ include exactly the same amount of information. It is easy to see that $T^{1,c}_{\mathscr{D}}$ is non-blocking and deterministic.

Other types of embeddings can also be defined for $\mathscr{D}$. For example, in an analysis problem, we might be interested in capturing all the possible runs of $\mathscr{D}$, and we are not interested in recording the controls. The one-step, control-abstract embedding of $\mathscr{D}$ is defined as a transition system $T^1_{\mathscr{D}}=(X,\delta,O,o)$ where X, O, and o are defined exactly as above. The transition function δ is defined as $x'=\delta(x)$ if and only if there exists $u\in U$ such that $x'=f(x,u)$.

Transition system $T^1_{\mathscr{D}}$ is a particular case of another timed, control-abstract embedding of $\mathscr{D}$. This embedding, which we will denote by $T^{\mathbb{N}}_{\mathscr{D}}=(X,\Sigma,\delta,O,o)$, is a transition system whose X, O, and o are inherited from $T^1_{\mathscr{D}}$ (and $T^{1,c}_{\mathscr{D}}$). Its set of inputs $\Sigma=\mathbb{N}$ and its transition function δ is defined as $x'=\delta(x,k)$ if and only if there exist $u_0,u_1,\ldots,u_{k-1}\in U$ driving system (4) from $x_0=x$ to $x'=x_k$. Finally, a time-abstract, control abstract embedding of $\mathscr{D}$, denoted simply by $T_{\mathscr{D}}=(X,\delta,O,o)$, is a transition system that can be defined by taking the transition function δ as $x'=\delta(x)$ if and only if there exist $k\in\mathbb{N}$ and $u_0,u_1,\ldots,u_{k-1}\in U$ driving system (4) from $x_0=x$ to $x'=x_k$.

It is important to note that our definition of an embedding transition system can accommodate more general situations in which the observations are not simply given as a function of the state, as is the case in Equation (4). There are several situations in which only a finite number of properties that the states can satisfy are of interest. Such properties, or predicates, can be given as a set of functions over the state space. For example, assume that we have a set of scalar

[1]Since the *Post* operator was defined for a set of inputs, the correct notation here is $Post_T(x,\{\sigma\})$. For simplicity, and with a slight abuse of notation, we omit the set notation when only a singleton input is considered. The same observation applies to the *Pre* operator.

[2]As before, without the risk of confusion, and to keep the notation simple, we slightly abuse the notation when we refer to singletons and sets made of just one element. Formally, δ outputs a set, while f outputs a singleton. In this case, the set produced by δ has just one element.

polynomials $\{p_1, p_2, \dots, p_L\}$ defined in $\mathbb{R}^n$. Each p_i defines a partition of $\mathbb{R}^n$ into three regions, which correspond to $p_i < 0$, $p_i = 0$, and $p_i > 0$, and which can be labeled simply as n, z, and p, respectively. If all L polynomials are considered, the state space is partitioned into 3^L regions that are labeled as $(p_1^e, p_2^e, \dots, p_L^e)$, with $p_i^e \in \{n, z, p\}$, $i = 1, 2, \dots, L$. If we are interested in how the trajectories of $\mathscr{D}$ move among these regions, the set of observations can be defined as

$$O = \{(p_1^e, p_2^e, \dots, p_L^e) \,|\, p_i^e \in \{n, z, p\}, i = 1, 2, \dots, L\}. \quad (5)$$

The observation map is given by:

$$o(x) = (p_1^e, p_2^e, \dots, p_L^e), \quad (6)$$

where $p_i^e = n$ if $p_i(x) < 0$, $p_i^e = z$ if $p_i(x) = 0$, and $p_i^e = p$ if $p_i(x) > 0$, for all $i = 1, 2, \dots, L$.

Alternatively, polynomials p_i, $i = 1, 2, \dots, L$ can be used to define a set of predicates, *e.g.,*

$$\pi_i : \ p_i < 0, \ i = 1, 2, \dots, L. \quad (7)$$

The set of observations is then the power set of the set of predicates

$$O = 2^{\{\pi_i, i=1,2,\dots,L\}} \quad (8)$$

and the observation map $o : X \to O$ is defined as

$$o(x) = \{\pi_i \,|\, p_i(x) < 0\} \quad (9)$$

Finally, in several practical applications, the observations are defined as labels $\{P_1, P_2, \dots, P_L\}$ for a set of possibly overlapping "regions" that cover the state space (*i.e.,* $\cup_{i=1,\dots,L} P_i = X$). In this case, with the observation set $O = 2^{\{P_1, P_2, \dots, P_L\}}$, the observation map $o : X \to O$ is defined as

$$o(x) = \{P_i \,|\, x \in P_i\} \quad (10)$$

Example 2: Consider a planar, discrete-time affine control system described by

$$\mathscr{D}_{lin} : \quad \begin{array}{rcl} x_{k+1} & = & Ax_k + Bu_k + b, \ k = 0, 1, 2, \dots, \\ y_k & = & Cx_k, \end{array} \quad (11)$$

where

$$A = \begin{bmatrix} 0.95 & -0.5 \\ 0.5 & 0.65 \end{bmatrix}, B = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 0.5 \\ -1.3 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

The one-step timed controlled embedding $T^{1,c}_{\mathscr{D}_{lin}} = (X, \Sigma, \delta, O, o)$ is defined by

- the set of states $X = \mathbb{R}^2$,
- the set of inputs $\Sigma = \mathbb{R}$,
- the transition function $\delta(x, u) = Ax + Bu + b$,
- the set of observations $O = \mathbb{R}$, and
- the observation map $o(x) = Cx$.

The input word $0, 0, 0, \dots$ determines the (autonomous) run

$$\begin{bmatrix} 8.0 \\ 5.0 \end{bmatrix}, \begin{bmatrix} 5.600 \\ 5.950 \end{bmatrix}, \begin{bmatrix} 2.8450 \\ 5.3675 \end{bmatrix}, \begin{bmatrix} 0.5190 \\ 3.6114 \end{bmatrix}, \begin{bmatrix} -0.8126 \\ 1.3069 \end{bmatrix},$$
$$\begin{bmatrix} -0.9255 \\ -0.8568 \end{bmatrix}, \begin{bmatrix} 0.0492 \\ -2.3197 \end{bmatrix}, \dots$$

shown in blue in Figure 2 (left) and the output word 8.0000, 5.6000, 2.8450, 0.5190, -0.8126, -0.9255, 0.0492, ….

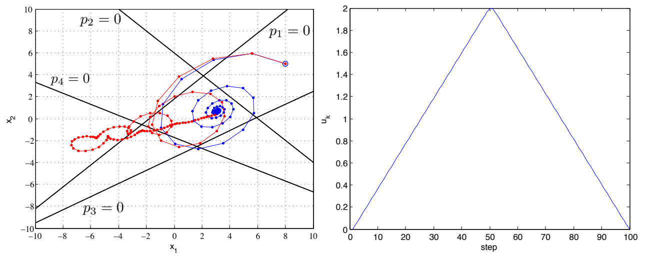


Fig. 2. Sample runs of the discrete-time system from Equation (11) are shown on the left: the blue run corresponds to the autonomous system ($u_k = 0$, $k = 0, 1, 2, \dots$), while the red run is produced by the input shown on the right.

The input word $u_0 u_1 u_2, \dots, u_{100}$ with $u_k = 0.04k$ for $k \le 50$ and $u_k = -0.04k + 4$ for $50 < k \le 100$, shown in Figure 2 (right), produces the run

$$\begin{bmatrix} 8.0 \\ 5.0 \end{bmatrix}, \begin{bmatrix} 5.600 \\ 5.950 \end{bmatrix}, \begin{bmatrix} 2.8042 \\ 5.4491 \end{bmatrix}, \begin{bmatrix} 0.3578 \\ 3.8073 \end{bmatrix}, \begin{bmatrix} -1.1862 \\ 1.5985 \end{bmatrix},$$
$$\begin{bmatrix} -1.5894 \\ -0.5275 \end{bmatrix}, \begin{bmatrix} -0.9503 \\ -2.0294 \end{bmatrix}, \dots$$

shown in red in Figure 2 (left) and the output word 8.0000, 5.6000, 2.8042, 0.3578, -1.1862, -1.5894, -0.9503, ….

Assume now that we are only interested in how the runs of the system behave with respect to a set of affine functions, *e.g.,*

$$\begin{array}{rcl} p_1(x) & = & [1 \ -1]^\top x + 1.8, \\ p_2(x) & = & [1 \, 1]^\top x - 6, \\ p_3(x) & = & [0.6 \ -1]^\top x - 3.5, \\ p_4(x) & = & [0.5 \, 1]^\top x + 1.7. \end{array}$$

If the approach described in Equations (5) and (6) is used, then the word generated by the blue run from the left of Figure 2 is

$$\begin{array}{l} (p,p,n,p), (p,p,n,p), (n,p,n,p), (n,n,n,p), \\ (n,n,n,p), (p,n,n,p), (p,n,n,n), (p,n,p,n), \dots \end{array} \quad (12)$$

The word generated by the red run coincides with the above word in the first 7 entries; the 8th entry is replaced by (p,n,n,n).

If the properties of interest are formulated in terms of a set of predicates, as in Equations (7), (8), and (9), then the word of the blue run is

$$\begin{array}{l} \{\pi_3\}, \{\pi_3\}, \{\pi_1, \pi_3\}, \{\pi_1, \pi_2, \pi_3\}, \{\pi_1, \pi_2, \pi_3\}, \\ \{\pi_2, \pi_3\}, \{\pi_2, \pi_3, \pi_4\}, \{\pi_2, \pi_4\}, \dots \end{array} \quad (13)$$

In the red run, the 8th entry is replaced by $\{\pi_2, \pi_3, \pi_4\}$.

III. Simulation and Bisimulation

A number of analysis and control techniques, such as the one presented in Section V, have been developed to handle only finite transition systems (*e.g.* when an explicit representation of all system states is required). Such these methods become computationally challenging as the size of the state set of the system increases, which limits the applicability of these methods due to the infamous "curse of dimensionality".

In this section, we introduce *finite abstractions*, which can be used to reduce the size of a finite system or map an infinite system to a finite one for the purpose of analysis and control.

Intuitively, an abstraction of a transition system T preserves some of its details required for analysis and control but ignores aspects that do not influence the results. More specifically, a state of the abstract model represents a large or infinite set of states in the original, *concrete* model that are somehow equivalent (*e.g.* all equivalent states might have the same observation). An abstraction could be equivalent to the concrete model with respect to the satisfaction of all specifications. Alternatively, it could provide an approximation, guaranteing that satisfaction of a specification in the abstract model implies satisfaction in the original system. Equivalent abstractions are based on the notion of *bisimulation*, while approximate abstractions are constructed using *simulation relations*.

The notions of simulation and bisimulation that we consider here are relations between systems and their quotients. Such relations can be defined, in general, between systems sharing the same sets of observations [2]. There also exist relaxed notions of bisimulations, such as weak [24], probabilistic [25], and approximate bisimulations [26], [27], which go beyond the scope of this paper.

The observation map o of a transition system $T = (X,\Sigma,\delta,O,o)$ induces an equivalence relation $\sim \subseteq X \times X$ over the set of states X of T.

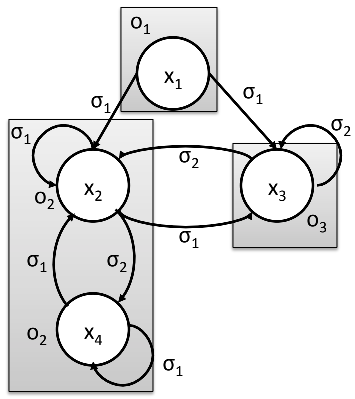
Definition 2 (Observational equivalence): States $x_1, x_2 \in X$ are observationally equivalent (written as $x_1 \sim x_2$) if and only if $o(x_1) = o(x_2)$.

Definition 3 (Quotient transition system): The observational equivalence relation $\sim$ naturally induces a *quotient transition system* $T/_\sim = (X/_\sim, \Sigma, \delta_\sim, O, o_\sim)$, where

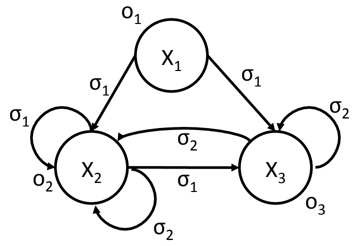
- the set of states $X/_\sim$ is the quotient space (*i.e.,* the set of all equivalence classes),
- the set of inputs Σ is inherited from the original system,
- the transition relation $\delta_\sim$ is defined as follows: for states $X_i, X_j \in X/_\sim$ and input $\sigma \in \Sigma$, we include transition $X_j \in \delta_\sim(X_i,\sigma)$ if and only if there exist states x_1 and x_2 of T in equivalence classes X_i and X_j, respectively, such that x_2 is reachable from x_1 in one step under input σ (*i.e.,* $x_2 \in \delta(x_1,\sigma)$),
- the set of observations O is inherited from T, and
- the observation $o_\sim(X)$ of a state $X_i \in X/_\sim$ is given by $o_\sim(X_i) = o(x)$ for all states x from equivalence class X_i.

Given an equivalence class $X_i \in X/_\sim$, we denote the set of all equivalent states of T in that class by $con(X_i) \subseteq X$, where *con* stands for *concretization map*. For a state X_i of $T/_\sim$ the set $con(X_i)$ is, in general, a region of T and if $\mathbb{X} \subseteq X/_\sim$ is a region of $T/_\sim$, then $con(\mathbb{X}) = \bigcup_{X_i \in \mathbb{X}} con(X_i)$ is a region of T. The observation map $o_\sim$ of $T/_\sim$ is well defined, since all states $x \in con(X_i)$ from an equivalence class $X_i \in X/_\sim$ have the same observation (*i.e.,* $\forall x \in con(X_i), o(x) = o_\sim(X_i)$).

Given states $X_i, X_j \in X/_\sim$, we can assign transitions in $T/_\sim$ through computation of the successor states (Equation (2)) of each equivalence class, *i.e.*, $X_j \in \delta_\sim(X_i,\sigma)$ if and only



(a) A finite, nondeterministic control transition system T.



(b) The quotient $T/_\sim$ of transition system T.

Fig. 3. The quotient $T/_\sim$ of a transition system T under the observational equivalence relation $\sim$. States that are equivalent in T (*i.e.,* states sharing the same observation) are highlighted (see Example 3 for additional details).

if

$$Post_T(con(X_i),\sigma) \cap con(X_j) \neq \emptyset. \tag{14}$$

Equivalently, we can assign transitions in $T/_\sim$ by computing the set of predecessors (Equation (3)) of each class: $X_j \in \delta_\sim(X_i,\sigma)$ if and only if

$$con(X_i) \cap Pre_T(con(X_j),\sigma) \neq \emptyset. \tag{15}$$

Example 3: Consider the transition system from Figure 1 shown again for convenience in Figure 3(a). The states with the same observations are equivalent. The corresponding equivalence classes and highlighted in Figure 3(a). The set of states of the quotient $T/_\sim$, shown in Figure 3(b), is $X/_\sim = \{X_1, X_2, X_3\}$. The set of all states of T in an equivalence class X_i is given by the concretization map $con()$: $con(X_1) = \{x_1\}$, $con(X_2) = \{x_2, x_4\}$, and $con(X_3) = \{x_3\}$. The observations of $T/_\sim$ are inherited from T. The transitions in $\delta_\sim$ are assigned as stated in Definition 3.

From Definition 3, it follows that for all states (equivalence classes) $X_i \in X/_\sim$ of $T/_\sim$, we have

$$\mathscr{L}_T(con(X_i)) \subseteq \mathscr{L}_{T/_\sim}(X_i). \tag{16}$$

In other words, the quotient control transition system $T/_\sim$ can produce any word w_O that can be produced by the original, concrete transition system T. However, in general there exists words in $\mathscr{L}_{T/_\sim}(X_i)$ that are *spurious* and do not represent valid behavior of T (*i.e.,* they are not part of $\mathscr{L}_T(con(X_i))$).

Since any behavior of T can be reproduced by $T/_\sim$, we say that $T/_\sim$ *simulates* T. As we will discuss later in the paper, this guarantees that, if a linear temporal logic formula is satisfied at some state X_i of $T/_\sim$, then the formula will be satisfied at all the states of T contained in $con(X_i)$.

Definition 4 (Bisimulation): The equivalence relation $\sim$ induced by the observation map o is a bisimulation of a transition system $T = (X, \Sigma, \delta, O, o)$ if, for all states $x_1, x_2 \in X$, and all inputs $\sigma \in \Sigma$, if $x_1 \sim x_2$ and $x_1' \in \delta(x_1, \sigma)$, then there exist $x_2' \in X$ such that $x_2' \in \delta(x_2, \sigma)$ and $x_1' \sim x_2'$.

If $\sim$ is a bisimulation, then the quotient transition system $T/_\sim$ is called a *bisimulation quotient* of T, and the transition systems T and $T/_\sim$ are called *bisimilar*. To explicitly distinguish between a simulation and a bisimulation, we sometimes denote the latter by $\approx$ and use $T/_\approx$ to denote a bisimulation quotient. In other words, the quotient $T/_\approx$ is the quotient $T/_\sim$ when Definition 4 is satisfied.

For example, in Figure 3(b), the quotient $T/_\sim$ is not a bisimulation of the transition system T shown in Figure 3(a). Note that x_2, which is in the equivalence class X_2 has a transition to some state in the equivalence class X_3 under input σ_1. However, x_4, which is also in the equivalence class X_2 does not have a transition to some state in the equivalence class X_3 under σ_1.

Definition 4 establishes bisimulation as a property of the quotient $T/_\sim$, when transitions originating at equivalent states in T satisfy certain conditions. As an immediate consequence of bisimulation, we can guarantee the language equivalence between the quotient $T/_\approx$ and the concrete system T. In other words, for all states $X_i \in X/_\approx$, it holds that

$$\mathscr{L}_T(con(X_i)) = \mathscr{L}_{T/_\approx}(X_i). \tag{17}$$

In Definition 4, we gave conditions on the transitions originating at equivalent states in system T, required for the observational equivalence relation $\sim$ to be a bisimulation. Following from Definition 4, a computationally attractive characterization of bisimulation can be given by considering the set of predecessors (Equation (3)) of each equivalence class.

Theorem 1 (Bisimulation characterization): The equivalence relation $\sim$ is a bisimulation for a transition system $T = (X, \Sigma, \delta, O, o)$ if and only if for all equivalence classes $X_i \in X/_\sim$ and for all inputs $\sigma \in \Sigma$, $Pre_T(con(X_i), \sigma)$ is either empty or a finite union of equivalence classes. Equivalently, the bisimulation property from Definition 4 is violated at state $X_i \in X/_\sim$ if there exist an input $\sigma \in \Sigma$ and a state $X_j \in X/_\sim$, such that

$$\emptyset \subset con(X_i) \cap Pre_T(con(X_j), \sigma) \subset con(X_i). \tag{18}$$

Equation (18) leads to an approach for the construction of the coarsest bisimulation $\approx$. Given a transition system T, the iterative procedure known as the "bisimulation algorithm" (summarized as Algorithm 1) starts with the observational equivalence relation $\sim$ and uses it to identify equivalence classes from $X/_\sim$ that satisfy Equation (18). Then, it iteratively refines these classes until the characterization from Theorem 1 is satisfied. This guarantees that the equivalence relation $\sim_r$ returned after the algorithm terminates is indeed $\approx$ — a bisimulation of T, which can be used to construct the bisimulation quotient $T/_\approx$.

Algorithm 1 $\approx$=BISIMULATION(T): Construct the coarsest observation-preserving bisimulation quotient $\approx$ of $T = (X, \Sigma, \delta, O, o)$

```
1: Initialize $\sim_r := \sim$
2: **while** there exist equivalence classes $X_i, X_j \in X/_{\sim_r}$ and $\sigma \in \Sigma$ such that
        $\emptyset \subset con(X_i) \cap Pre_T(con(X_j), \sigma) \subset con(X_i)$ **do**
3:     Construct equivalence class $X_1$ such that $con(X_1) := con(X_i) \cap Pre_T(con(X_j), \sigma)$
4:     Construct equivalence class $X_2$ such that $con(X_2) := con(X_i) \setminus Pre_T(con(X_j), \sigma)$
5:     $X/_{\sim_r} := X/_{\sim_r} \setminus \{X_i\} \bigcup \{X_1, X_2\}$
6: **end while**
7: return $\sim_r$ ($\sim_r = \approx$)
```

IV. Temporal Logics and Automata

In this paper, we consider control specifications given as formulas of a particular type of temporal logic, called Linear Temporal Logic (LTL). Such formulas are expressive enough to capture a rich spectrum of properties, including safety (nothing bad will ever happen), liveness (something good will eventually happen), and more complex combinations of Boolean and temporal statements. In this section, we introduce the syntax and semantics of LTL and we illustrate them through several examples. We also define the automata that will be later used for system control from such specifications.

Temporal logics were originally developed by philosophers to reason about how truth and knowledge change over time. They were later adapted in computer science and used to specify the correctness of digital circuits and computer programs. Besides several more expressive temporal logics, LTL, Computation Tree Logic (CTL) and their unifying CTL* framework [28], [3] are the most commonly encountered.

Most temporal logics, including LTL, have probabilistic versions. In particular, the probabilistic version of LTL, called Probabilistic LTL (PLTL), is simply defined by adding a probability operator that quantifies the satisfaction probability in front of the formula. Its semantics is defined over a Markov decision process (MDP), the probabilistic version of the transition system defined in Section II. Probabilistic temporal logics go beyond the scope of this book, and the interested reader is referred to [19], [29], [30]. There also exist logics, such as Bounded Linear Temporal Logic (BLTL) [31], Signal Temporal Logic (STL) [32], and Metric

Temporal Logic (MTL) [33], in which the temporal operators have specific time intervals.

A. Linear Temporal Logic

Linear Temporal Logic (LTL) formulas are constructed from a set of observations, Boolean operators, and temporal operators. We use the standard notation for the Boolean operators (*i.e.,* $\top$ (true), $\neg$ (negation), $\wedge$ (conjunction)) and the graphical notation for the temporal operators (*e.g.,* $\bigcirc$ ("next"), U ("until")). The $\bigcirc$ operator is a unary prefix operator and is followed by a single LTL formula, while U is a binary infix operator. Formally, we define the syntax of LTL formulas as follows:

Definition 5 (LTL Syntax): A (propositional) Linear Temporal Logic (LTL) formula ϕ over a given set of observations O is recursively defined as

$$\phi = \top \mid o \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 U \phi_2, \tag{19}$$

where $o \in O$ is an observation and ϕ, ϕ_1 and ϕ_2 are LTL formulas.

Unary operators have a higher precedence than binary ones and $\neg$ and $\bigcirc$ bind equally strong. The temporal operator U takes precedence over $\neg$ and $\wedge$ and is right-associative (*e.g.,* $\phi_1 U \phi_2 U \phi_3$ stands for $\phi_1 U(\phi_2 U \phi_3)$).

To obtain the full expressivity of propositional logic, additional operators are defined as

$$\begin{aligned} \phi_1 \vee \phi_2 &:= \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \phi_1 \rightarrow \phi_2 &:= \neg\phi_1 \vee \phi_2 \\ \phi_1 \leftrightarrow \phi_2 &:= (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1) \end{aligned}$$

In addition, the temporal operators $\Diamond$ ("eventually") and $\Box$ ("always") are defined as follows:

$$\begin{aligned} \Diamond\phi &:= \top U \phi \\ \Box\phi &:= \neg\Diamond\neg\phi \end{aligned}$$

By combining the various temporal operators, more complicated expressions can be obtained. For example, we will frequently use the combinations $\Diamond\Box$ ("eventually always") and $\Box\Diamond$ ("always eventually").

LTL formulas are interpreted over infinite words made of observations from O, *i.e.*, over O^ω. Formally, the LTL semantics are defined as follows:

Definition 6 (LTL Semantics): The satisfaction of formula ϕ over a set of observations O at position $k \in \mathbb{N}$, $k \geq 1$ by word $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$, denoted by $w_O(k) \vDash \phi$, is defined recursively as follows:

- $w_O(k) \vDash \top$,
- $w_O(k) \vDash o$ for some $o \in O$ if $w_O(k) = o$,
- $w_O(k) \vDash \neg\phi$ if $w_O(k) \nvDash \phi$,
- $w_O(k) \vDash \phi_1 \wedge \phi_2$ if $w_O(k) \vDash \phi_1$ and $w_O(k) \vDash \phi_2$,
- $w_O(k) \vDash \bigcirc\phi$ if $w_O(k+1) \vDash \phi$,
- $w_O(k) \vDash \phi_1 U \phi_2$ if there exist $j \geq k$ such that $w_O(j) \vDash \phi_2$ and, for all $k \leq i < j$, we have $w_O(i) \vDash \phi_1$.

A word w_O satisfies an LTL formula ϕ, written as $w_O \vDash \phi$, if $w_O(1) \vDash \phi$. We denote the language of infinite words that satisfy formula ϕ by $\mathscr{L}_\phi$.

In the following, we give an informal interpretation of the satisfaction of some frequently used LTL formulas.

- $\bigcirc\phi$ is satisfied at the current step if ϕ is satisfied at the next step.
- $\phi_1 U \phi_2$ is satisfied if ϕ_1 is satisfied "until" ϕ_2 becomes satisfied,
- $\Box\phi$ is satisfied if ϕ is satisfied at each step (*i.e.,* ϕ is "always" satisfied).
- $\Box\neg\phi$ is satisfied if $\neg\phi$ is satisfied at each step (*i.e.,* ϕ is "never" satisfied).
- $\Diamond\phi$ is satisfied if ϕ is satisfied at some future step (*i.e.,* ϕ is "eventually" satisfied).
- $\Diamond\Box\phi$ is satisfied if ϕ becomes satisfied at some future step and remains satisfied for all following steps (*i.e.,* ϕ is satisfied "eventually forever").
- Formula $\Box\Diamond\phi$ is satisfied if ϕ always becomes satisfied at some future step (*i.e.,* ϕ is satisfied "infinitely often").

B. Automata

We will use automata that accept languages satisfying LTL formulas over the set of observations O. There is, therefore, no coincidence that the input alphabets of the automata defined below is O.

Definition 7 (Büchi automaton): A (nondeterministic) Büchi automaton is a tuple $B = (S, S_0, O, \delta, F)$, where

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- O is the input alphabet,
- $\delta : S \times O \rightarrow 2^S$ is a nondeterministic transition function, and
- $F \subseteq S$ is the set of accepting (final) states.

A Büchi automaton is deterministic if S_0 is a singleton and $\delta(s, o)$ is either $\emptyset$ or a singleton for all $s \in S$ and $o \in O$. The semantics of a Büchi automaton are defined over infinite input words in O^ω. A run of B over a word $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$ is a sequence $w_S = w_S(1)w_S(2)w_S(3)\ldots \in S^\omega$ where $w_S(1) \in S_0$ and $w_S(k+1) \in \delta(w_S(k), w_O(k))$ for all $k \geq 1$.

Definition 8 (Büchi acceptance): Let $\inf(w_S)$ denote the set of states that appear in the run w_S infinitely often. An input word w_O is accepted by B if and only if there exists at least one run w_S over w_O that visits F infinitely often, *i.e.*, $\inf(w_S) \cap F \neq \emptyset$.

We denote by $\mathscr{L}_B$ the language accepted by B, *i.e.,* the set of all words accepted by B. An LTL formula ϕ over a set O can always be translated into a Büchi automaton B_ϕ with input alphabet O and $\mathscr{O}(|\phi| \cdot 2^{|\phi|})$ states ($|\phi|$ denotes the length of ϕ, which is defined as the total number of occurrences of observations and operators) that accepts all and only words satisfying ϕ (*i.e.,* $\mathscr{L}_{B_\phi} = \mathscr{L}_\phi$). This translation can be performed using efficient, off-the-shelf software tools, such as LTL2BA.

Note that, in general, a nondeterministic Büchi automaton is obtained by translating an LTL formula. While certain Büchi automata can be determinized, a sound and complete procedure for determinizing general Büchi automata does not exist and, in fact, there exist LTL formulas which cannot be converted to deterministic Büchi automata.

Example 4: Examples of Büchi automata for some commonly encountered LTL formulas are shown in Figure 4. Even when a nondeterministic Büchi automaton was obtained through the translation with LTL2BA tool, the automaton was simplified and determinized by hand whenever possible (*e.g.,* for formulas ϕ_1, ϕ_4 and ϕ_5). Even so, some of the automata, such as the ones obtained for LTL formulas ϕ_3, ϕ_6 and ϕ_7 cannot be determinized. In fact, it is known that formulas of the type $\Diamond\Box\phi$ cannot be converted to a deterministic Büchi automaton, which can be verified for ϕ_3. By simple inspection of the automaton in Figure 4 (c), it can be seen that if o_1 was removed from the self transition at s_0, which would make the automaton deterministic, then word $o_1 o_2 (o_1)^\omega$, which is obviously satisfying, would not be accepted. To understand why formulas ϕ_6 and ϕ_7 result in nondeterministic Büchi automata, we can rewrite them as

$$\begin{aligned}
\phi_6 &= \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2 = \Box\Diamond o_1 \wedge \Diamond\Box\neg o_2 && (20)\\
\phi_7 &= \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2 = && (21)\\
&= (\Box\Diamond o_1 \wedge \Box\Diamond o_2) \vee \neg\Box\Diamond o_1 = && (22)\\
&= (\Box\Diamond o_1 \wedge \Box\Diamond o_2) \vee \Diamond\Box\neg o_1 && (23)
\end{aligned}$$

to reveal that both contain a $\Diamond\Box$ sub-formula.

Definition 9 (Rabin automaton): A (nondeterministic) Rabin automaton is a tuple $R = (S, S_0, O, \delta, F)$, where

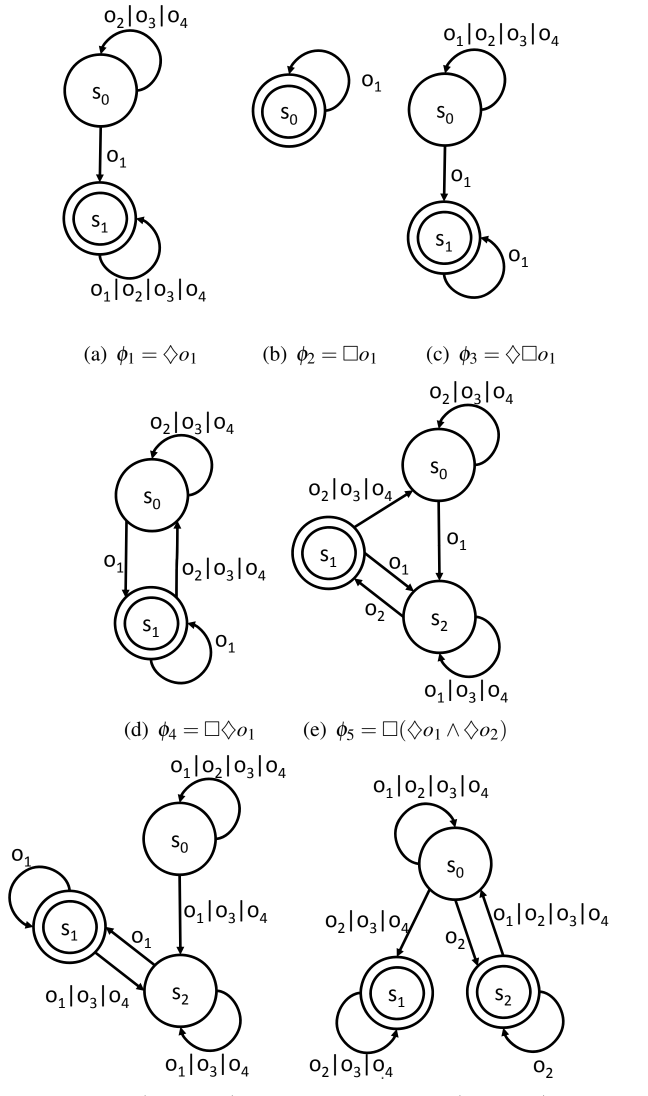
- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- O is the input alphabet,
- $\delta : S \times O \to 2^S$ is a transition map, and
- $F = \{(G_1, B_1), \ldots, (G_n, B_n)\}$, where $G_i, B_i \subseteq S$, $i = 1, 2, \ldots, n$ is the acceptance condition.

A Rabin automaton R is deterministic if S_0 is a singleton and $\delta(s, o)$ is either $\emptyset$ or a singleton, for all $s \in S$ and $o \in O$. The semantics of a Rabin automaton are defined over infinite input words in O^ω. A run of R over a word $w_O = w_O(1) w_O(2) w_O(3) \ldots \in O^\omega$ is a sequence $w_S = w_S(1) w_S(2) w_S(3) \ldots \in S^\omega$, where $w_S(1) \in S_0$ and $w_S(k+1) \in \delta(w_S(k), w_O(k))$ for all $k \geq 1$.

Definition 10 (Rabin acceptance): Let $\inf(w_S)$ denote the set of states that appear in the run w_S infinitely often. A run w_S is accepted by R if $\inf(w_S) \cap G_i \neq \emptyset \wedge \inf(w_S) \cap B_i = \emptyset$ for some $i \in \{1, \ldots, n\}$. An input word w_O is accepted by a Rabin automaton R if some run over w_O is accepted by R.

We denote by $\mathscr{L}_R$ the language accepted by R, *i.e.*, the set of all words accepted by R. Given an LTL formula ϕ, one can build a deterministic Rabin automaton R with input alphabet O, $2^{2^{\mathscr{O}(|\phi| \cdot \log|\phi|)}}$ states, and $2^{\mathscr{O}(|\phi|)}$ pairs in its acceptance condition, such that $\mathscr{L}_R = \mathscr{L}_\phi$. The translation can be done using off-the-shelf software tools. Note that a Büchi automaton B is a Rabin automaton R with one pair in its acceptance condition $F_R = \{(G, B)\}$ where $G = F_B$ and $B = \emptyset$.

Example 5: Even though LTL formulas ϕ_3, ϕ_6 and ϕ_7 could only be translated into nondeterministic Büchi automata in Example 4, we can translate them instead into the deterministic Rabin automata shown in Figure 5. The Rabin automata for formulas ϕ_3 and ϕ_6 contain only a single pair in their acceptance conditions, while the one for ϕ_7 contains two pairs.



(a) $\phi_1 = \Diamond o_1$ (b) $\phi_2 = \Box o_1$ (c) $\phi_3 = \Diamond\Box o_1$

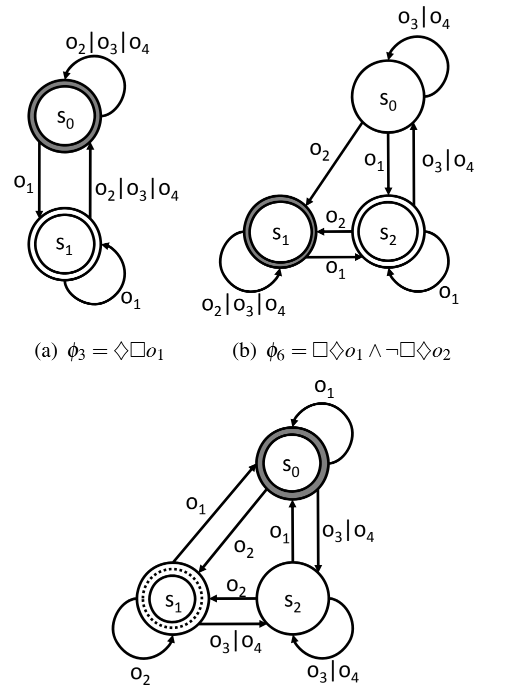
(d) $\phi_4 = \Box\Diamond o_1$ (e) $\phi_5 = \Box(\Diamond o_1 \wedge \Diamond o_2)$

(f) $\phi_6 = \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2$ (g) $\phi_7 = \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2$

Fig. 4. Graphical representation of the Büchi automata for some commonly used LTL formulas over the set of observations $O = \{o_1, \ldots, o_4\}$. For all automata, s_0 is the initial state and the final states are indicated by double circles. For simplicity of the representation if several transitions are present between two states, only one transition labeled by the set of all inputs (separated by the symbol |) labeling all transitions is shown. For additional details, see Example 4.

V. Temporal Logic Control for Finite Transition Systems

In this section, we treat the general problem of controlling non-deterministic finite transition systems from specifications given as LTL formulas over their sets of observations. We show that, in the most general case, the problem can be reduced to a Rabin game [34]. There are various approaches to solve Rabin games [35], [36], [37]. The solution we present is based on [37]. The Rabin game based approach to



(a) $\phi_3 = \Diamond\Box o_1$ (b) $\phi_6 = \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2$

(c) $\phi_7 = \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2$

Fig. 5. Graphical representation of the Rabin automata for the LTL formulas from Example 4 resulting in nondeterministic Büchi automata. For each automaton, s_0 is the initial state. For the automata accepting formulas ϕ_3 and ϕ_6 the acceptance condition F is defined by one pair of singletons (G,B) where $G = \{s_1\}, B = \{s_0\}$ in (a) and $G = \{s_2\}, B = \{s_1\}$ in (b) (good and bad states are denoted by unshaded or shaded double circles, respectively). For the automaton accepting ϕ_7 the acceptance condition includes two pairs where $G_1 = \{s_1, s_2\}, B_1 = \{s_0\}$ and $G_2 = \{s_1\}, B_2 = \emptyset$ (the single bad state is denoted by a shaded circle and the good state that is common for both pairs of the acceptance conditions is denoted by a solid and dashed circle in (c)). As in Figure 4, for simplicity of the representation if several transitions are present between two states, only one transition labeled by the set of all inputs (separated by the symbol |) labeling all transitions is shown. For additional details, see Example 5.

the control problem from this section is based on [38]. For the particular case when the LTL formula can be translated to a deterministic Büchi automaton, we show that the control problem reduced to a Büchi game [39], for which efficient solutions exist [40]. A treatment of the control problem for this case can be found in [41]. In both cases, the control strategy for the original transition system takes the form of a feedback control automaton, which is easy to interpret and implement.

For simplicity of exposition, we only consider synthesis from LTL specifications. Readers interested in CTL and CTL* specifications are referred to [42], [43],[44]. There has also been some interest in combining optimality with correctness in formal synthesis. Examples include optimal LTL control for transition systems [45], [46], [47] and Markov decision processes [48], [49], [50].

Definition 11 (Control strategy): A (history dependent) *control function* $\Omega : X^+ \to \Sigma$ for control transition system $T = (X, \Sigma, \delta, O, o)$ maps a finite, nonempty sequence of states to an input of T. A control function Ω and a set of initial states $X_0 \subseteq X$ provide a *control strategy* for T.

We denote a control strategy by (X_0, Ω), the set of all trajectories of the closed loop system T under the control strategy by $T(X_0, \Omega)$, and the set of all words produced by the closed loop T as $\mathscr{L}_T(X_0, \Omega)$. For any trajectory $x_1x_2x_3 \ldots \in T(X_0, \Omega)$ we have $x_1 \in X_0$ and $x_{k+1} \in \delta(x_k, \sigma_k)$, where $\sigma_k = \Omega(x_1, \ldots, x_k)$, for all $k \geq 1$.

Definition 12 (Largest Controlled Satisfying Region): Given a transition system $T = (X, \Sigma, \delta, O, o)$ and an LTL formula ϕ over O, the largest controlled satisfying region $X_T^\phi \subseteq X$ is the largest set of states for which there exists a control function $\Omega : X^+ \to \Sigma$ such that all trajectories $T(X_T^\phi, \Omega)$ of the closed loop system satisfy ϕ (*i.e.,* $\mathscr{L}_T(X_T^\phi, \Omega) \subseteq \mathscr{L}_\phi$).

The LTL control problem that we consider in this section can be formulated as:

Problem 1 (Largest Controlled Satisfying Region Problem): Given a finite transition system $T = (X, \Sigma, \delta, O, o)$ and an LTL formula ϕ over O, find a control strategy (X_T^ϕ, Ω) such that X_T^ϕ is the largest controlled satisfying region and $\mathscr{L}_T(X_T^\phi, \Omega) \subseteq \mathscr{L}_\phi$.

The control problem for transition systems from LTL specifications is stated in most general form in Problem 1, *i.e.,* for nondeterministic transition systems and full LTL specifications. In the following section, we present an algorithm to solve this problem and discuss the related complexity. In the presented algorithm, the control synthesis problem is treated as a game played on a finite graph and approached using automata theoretic methods. Such game semantics are introduced due to the nondeterminism of the transition system and the accepting condition of a Rabin automaton. However, if the transition system is deterministic, the control problem can be solved through model checking techniques in a more efficient way. In Section V-B, we focus on a particular case of this problem, *e.g.,* when the LTL formula can be translated to a deterministic Büchi automaton (a dLTL specification).

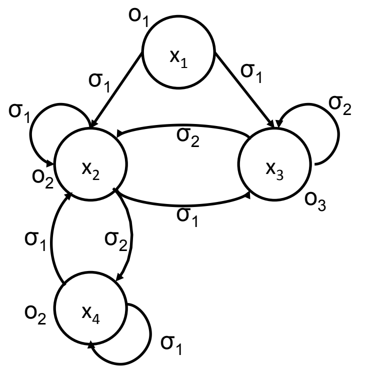
A. Control of Transition Systems from LTL Specifications

In this section, we provide a solution to the general problem of controlling finite, nondeterministic systems from LTL specifications (Problem 1). The procedure involves the translation of the LTL formula into a deterministic Rabin automaton, the construction of the product automaton of the transition system and the Rabin automaton, followed by the solution of a Rabin game on this product. The solution of the Rabin game is a control strategy for the product automaton, and finally this solution is transformed into a control strategy for the transition system. The resulting control strategy takes the form of a *feedback control automaton*, which reads the current state of T and produces the control input to be applied at that state. The overall control procedure is summarized in Algorithm 2. In the rest of this section, we provide the details of this procedure.

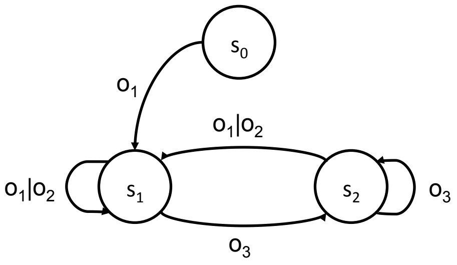
Step 1: Construction of the Rabin Automaton

Algorithm 2 LTL CONTROL(T,ϕ) : Control strategy (X_T^{ϕ},Ω) such that all trajectories in $T(X_T^{\phi},\Omega)$ satisfy ϕ

```
1: Translate $\phi$ into a deterministic Rabin automaton $R = (S,S_0,O,\delta_R,F)$.
2: Build a product automaton $P = T \otimes R$
3: Transform $P$ into a Rabin game
4: Solve the Rabin game
5: Map the solution to the Rabin game into a control strategy for the original transition system $T$
```



(a) Transition system T



(b) $\phi = o_1 \wedge (\diamond\Box(o_1 \vee o_2) \vee \diamond\Box o_3)$

Fig. 6. Graphical representations of transition system (a) and the Rabin automaton (b) from Example 6. For the automaton, s_0 is the initial state and the acceptance condition is defined by $F = \{(G_1,B_1),(G_2,B_2)\}$, where $G_1 = B_2 = \{s_2\}$ and $B_1 = G_2 = \{s_1\}$.

The first step is to translate the LTL specification ϕ into a deterministic Rabin automaton R. Note that there are readily available off-the-shelf tools for such translations.

Example 6: Consider the nondeterministic transition system $T = (X,\Sigma,\delta,O,o)$ from Example 1 shown in Figure 1, and reproduced for convenience in Figure 6(a). We consider the following specification "a trajectory of T originates at a state where o_1 is satisfied, and it eventually reaches and remains in a region where either o_1 or o_2 are satisfied, or o_3 is satisfied". The specification is formally defined as the LTL formula

$$\phi = o_1 \wedge (\diamond\Box(o_1 \vee o_2) \vee \diamond\Box o_3).$$

A Rabin automaton representation of the formula ϕ is shown in Figure 6(b).

Step 2: Construction of the Product Automaton

The second step is the construction of a product automaton between the transition system T and the Rabin automaton R, which is formally defined as:

Definition 13 (Controlled Rabin Product Automaton): The *controlled Rabin product automaton* $P = T \otimes R$ of a finite (control) transition system $T = (X,\Sigma,\delta,O,o)$ and a Rabin automaton $R = (S,S_0,O,\delta_R,F)$ is defined as $P = (S_P,S_{P0},\Sigma,\delta_P,F_P)$, where

- $S_P = X \times S$ is the set of states,
- $S_{P0} = X \times S_0$ is the set of initial states,
- Σ is the input alphabet,
- $\delta_P : S_P \times \Sigma \to 2^{S_P}$ is the transition map, where $\delta_P((x,s),\sigma) = \{(x',s') \in S_P \mid x' \in \delta(x,\sigma)$, and $s' = \delta_R(s,o(x))\}$, and
- $F_P = \{(X \times G_1, X \times B_1),\ldots,(X \times G_n, X \times B_n)\}$ is the Rabin acceptance condition.

This product automaton is a nondeterministic Rabin automaton with the same input alphabet Σ as T. Each accepting run $(x_1,s_1)(x_2,s_2)(x_3,s_3)\ldots$ of a product automaton $P = T \otimes R$ can be projected into a trajectory $x_1x_2x_3\ldots$ of T, such that the word $o(x_1)o(x_2)o(x_3)\ldots$ is accepted by R (*i.e.*, satisfies ϕ) and vice versa. This allows us to reduce Problem 1 to finding a control strategy for P. We define a control strategy for a Rabin automaton, and therefore for a product automaton constructed as in Definition 13, similarly as for a transition system. However, instead of history dependent control strategy, we introduce a memoryless strategy. As we will present later in this section, control strategies obtained by solving Rabin games (step 4 of Algorithm 2) are memoryless.

Definition 14 (Control strategy for a Rabin automaton): A memoryless *control function* $\pi : S \to O$ for a Rabin automaton $R = (S,S_0,O,\delta_R,F)$ maps a state of R to an input of R. A control function π and a set of initial states $W_0 \subseteq S_0$ provide a *control strategy* (W_0,π) for R.

A run $s_1s_2s_3\ldots$ *under strategy* (W_0,π) is a run satisfying the following two conditions: (1) $s_1 \in W_0$ and (2) $s_{k+1} \in \delta_R(s_k,\pi(s_k))$, for all $k \geq 1$.

The product automaton P allows us to reduce Problem 1 to the following problem:

Problem 2: Given a controlled Rabin product automaton $P = (S_P,S_{P0},\Sigma,\delta_P,F_P)$ find the largest set of initial states $W_{P0} \subseteq S_{P0}$ for which there exists a control function $\pi_P : S_P \to \Sigma$ such that each run of P under the strategy (W_{P0},π_P) satisfies the Rabin acceptance condition F_P.

Example 7: The product automaton $P = (S_P,S_{P0},\Sigma,\delta_P,F_P)$ of the transition system and the Rabin automaton from Example 6 (Figure 6) is shown in Figure 7. Note that the blocking states that are not reachable from the non-blocking initial state $p_0 = (x_1,s_0)$ are removed from P and are not shown in Figure 7.

Step 3: Translation to a Rabin Game

A Rabin game consists of a finite graph (V,E) containing a token. The token is always present in one of the states and can move along the edges. There are two players: a protagonist and an adversary. V is partitioned into protagonist's states $V_{\mathbf{P}}$ and adversary's states $V_{\mathbf{A}}$. The owner of the state

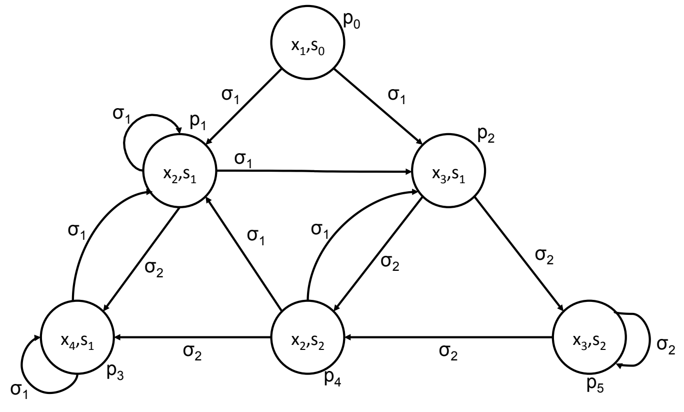


Fig. 7. Graphical representation of the product between the transition system from Figure 6(a) and the Rabin automaton from Figure 6(b). The initial state is $p_0 = (x_1, s_0)$. The accepting condition is defined by $F_P = \{(G_1, B_1), (G_2, B_2)\}$, where $G_1 = B_2 = \{p_4, p_5\}$ and $G_2 = B_1 = \{p_1, p_2, p_3\}$.

containing a token chooses the edge along which the token moves. A Rabin game is formally defined as:

Definition 15 (Rabin Game): A Rabin game played by two players (a protagonist and an adversary) on a graph (V,E) is a tuple $\mathbf{G} = (V_\mathbf{P}, V_\mathbf{A}, E, F_\mathbf{G})$, where

- $V_\mathbf{P}$ is the set of protagonist's states,
- $V_\mathbf{A}$ is the set of adversary's states,
- $V_\mathbf{P} \cup V_\mathbf{A} = V$, $V_\mathbf{P} \cap V_\mathbf{A} = \emptyset$,
- $E \subseteq V \times V$ is the set of possible actions,
- $F_\mathbf{G} = \{(G_1, B_1), \ldots, (G_n, B_n)\}$ is the winning condition for the protagonist, where $G_i, B_i \subseteq V$ for all $i \in \{1, \ldots, n\}$.

A play p is an infinite sequence of states visited by the token. Each play is winning either for the protagonist or the adversary. The protagonist wins if $inf(p) \cap G_i \neq \emptyset \wedge inf(p) \cap B_i = \emptyset$ for some $i \in \{1, \ldots, n\}$, where $\inf(p)$ denotes the set of states that appear in the play p infinitely often. The adversary wins in the rest of the cases. The winning region for the protagonist is defined as the set of states $W_\mathbf{P} \subseteq V$ such that there exists a control function $\pi_P : W_\mathbf{P} \cap V_\mathbf{P} \to E$, and all plays starting in the winning region and respecting the winning strategy are winning for the protagonist regardless of the adversary's choices. A solution to a Rabin game is a winning region and winning strategy for the protagonist.

The third step of Algorithm 2 is the construction of a Rabin game from the product automaton, which is performed as follows.

Definition 16 (Rabin game of a Rabin automaton): A *Rabin game* $\mathbf{G} = (V_\mathbf{P}, V_\mathbf{A}, E, F_\mathbf{G})$ of a Rabin automaton $P = (S_P, S_{P0}, \Sigma, \delta_P, F_P)$ is defined as:

- $V_\mathbf{P} = S_P$ is the protagonist's states,
- $V_\mathbf{A} = S_P \times \Sigma$ is the adversary's states,
- $E \subseteq \{V_\mathbf{P} \times V_\mathbf{A} \cup V_\mathbf{A} \times V_\mathbf{P}\}$ is the set of edges, which is defined as
  - $(q_\mathbf{P}, q_\mathbf{A}) \in E$ if $q_\mathbf{P} \in V_\mathbf{P}$, $q_\mathbf{A} \in V_\mathbf{A}$, and $q_\mathbf{A} = (q_\mathbf{P}, \sigma)$, where $\sigma \in \Sigma^{q_\mathbf{P}}$ (*i.e.,* if $\delta_P(q_\mathbf{P}, \sigma) \neq \emptyset$),
  - $(q_\mathbf{A}, q_\mathbf{P}) \in E$ if $q_\mathbf{A} \in V_\mathbf{A}$, $q_\mathbf{P} \in V_\mathbf{P}$, and $q_\mathbf{A} = (q'_\mathbf{P}, \sigma)$, and $q_\mathbf{P} \in \delta_P(q'_\mathbf{P}, \sigma)$,

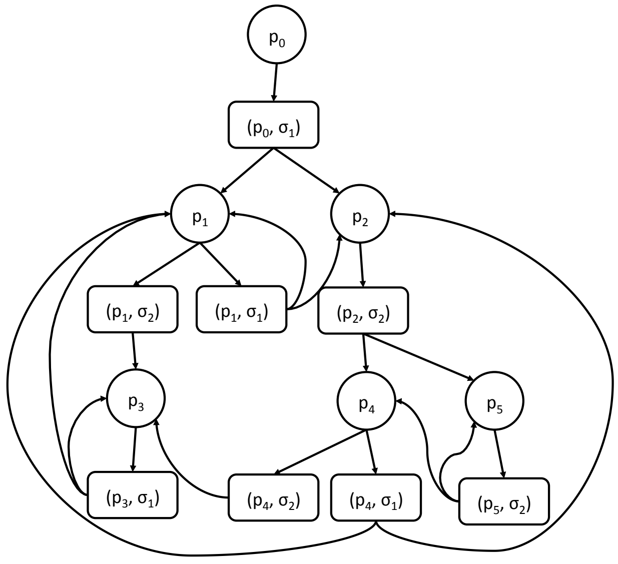


Fig. 8. Graphical representation of the Rabin game constructed from the Rabin automaton from Figure 7. An example play is $p = p_0(p_0, \sigma_1)p_2(p_2, \sigma_2)(p_5(p_5, \sigma_2))^\omega$.

- $F_\mathbf{G} = F_P$ is the protagonist's winning condition.

Intuitively, the protagonist chooses action σ, whereas the adversary resolves nondeterminism. Note that in a Rabin game constructed from a Rabin automaton, the protagonist's (adversary's) states can be reached in one step only from the adversary's (protagonist's) states. We will show later in this section that a solution to the Rabin game $\mathbf{G}$ can be easily transformed into a solution to Problem 2.

Example 8: The Rabin game of the product automaton from Example 7 is shown in Figure 8, where protagonist's states are represented as circles and adversary's states are represented as rectangles.

Step 4: Solving the Rabin Game

We present Horn's algorithm for solving Rabin games. The main idea behind the algorithm is as follows. The protagonist wins if he can infinitely often visit G_i and avoid B_i for some $i \in \{1, \ldots, n\}$. Conversely, the protagonist can not win if the adversary can infinitely often visit B_i for each $i \in \{1, \ldots, n\}$. Since it is sufficient for the protagonist to satisfy one of the conditions (G_i, B_i) from $F_\mathbf{G}$, the protagonist chooses a condition and tries to avoid visits to B_i and enforce visits to G_i. In turn the adversary tries to avoid G_i. By removing the states where the protagonist (or the adversary) can enforce a visit to a desired set, a smaller game is defined and the algorithm is applied to this game recursively. If the computation ends favorably for the adversary, then the protagonist chooses a different condition (G_j, B_j) from $F_\mathbf{G}$ and tries to win the game by satisfying this condition. For a given set $V' \subset V$, the set of states from which the protagonist (or the adversary) can enforce a visit to V' is called an *attractor set*, which is formally defined as follows:

Definition 17 (Protagonist's direct attractor): The protagonist's direct attractor of a set of states V', denoted by $\mathsf{A}^1_\mathbf{P}(V')$, is the set of all states $v_\mathbf{P} \in V_\mathbf{P}$, such that there exists an edge $(v_{\mathbf{P}}, v_{\mathbf{A}})$, where $v_{\mathbf{A}} \in V'$ together with the set of all states $v_{\mathbf{A}} \in V_{\mathbf{A}}$, such that for all $v_{\mathbf{P}} \in V_{\mathbf{P}}$ it holds that $(v_{\mathbf{A}}, v_{\mathbf{P}}) \in E$ implies $v_{\mathbf{P}} \in V'$:

$$\mathsf{A}^1_{\mathbf{P}}(V') := \{v_{\mathbf{P}} \in V_{\mathbf{P}} | (v_{\mathbf{P}}, v_{\mathbf{A}}) \in E, v_{\mathbf{A}} \in V'\} \bigcup$$

$$\{v_{\mathbf{A}} \in V_{\mathbf{A}} | \{v_{\mathbf{P}} | (v_{\mathbf{A}}, v_{\mathbf{P}}) \in E\} \subseteq V'\}.$$

Definition 18 (Adversary's direct attractor): The adversary's direct attractor of V', denoted by $\mathsf{A}^1_{\mathbf{A}}(V')$, is the set of all states $v_{\mathbf{A}} \in V_{\mathbf{A}}$, such that there exists an edge $(v_{\mathbf{A}}, v_{\mathbf{P}})$, where $v_{\mathbf{P}} \in V'$ together with the set of all states $v_{\mathbf{P}} \in V_{\mathbf{P}}$, such that for all $v_{\mathbf{A}} \in V_{\mathbf{A}}$ it holds that $(v_{\mathbf{P}}, v_{\mathbf{A}}) \in E$ implies $v_{\mathbf{A}} \in V'$:

$$\mathsf{A}^1_{\mathbf{A}}(V') := \{v_{\mathbf{A}} \in V_{\mathbf{A}} | (v_{\mathbf{A}}, v_{\mathbf{P}}) \in E, v_{\mathbf{P}} \in V'\} \bigcup$$

$$\{v_{\mathbf{P}} \in V_{\mathbf{P}} | \{v_{\mathbf{A}} | (v_{\mathbf{P}}, v_{\mathbf{A}}) \in E\} \subseteq V'\}.$$

In other words, the protagonist can enforce a visit to V' from each state $v \in \mathsf{A}^1_{\mathbf{P}}(V')$, regardless of the adversary's choice. Similarly, the adversary can enforce a visit to V' from each state $v \in \mathsf{A}^1_{\mathbf{A}}(V')$, regardless of the protagonist's choice.

Example 9: Consider the Rabin game shown in Figure 8. The protagonist's direct attractor set of $\{p_5\}$, $\mathsf{A}^1_{\mathbf{P}}(\{p_5\})$ is empty, since p_5 can be reached from (p_2, σ_2) and (p_5, σ_2), and for both these states the adversary can choose an edge incident to p_4 instead of p_5. On the other hand

$$\mathsf{A}^1_{\mathbf{P}}(\{p_4, p_5\}) = \{(p_2, \sigma_2), (p_5, \sigma_2)\},$$

since at (p_2, σ_2) (and similarly at (p_5, σ_2)), the adversary can either choose the edge $((p_2, \sigma_2), p_4)$ or $((p_2, \sigma_2), p_5)$ and both lead to $\{p_4, p_5\}$.

The adversary's direct attractor set of $\{p_5\}$ is $\mathsf{A}^1_{\mathbf{A}}(\{p_5\}) = \{(p_2, \sigma_2), (p_5, \sigma_2)\}$, since the adversary can enforce a visit to $\{p_5\}$ only from (p_2, σ_2) and (p_5, σ_2). As there are no other adversary states that have an edge to a state from the set $\{p_4, p_5\}$, we have:

$$\mathsf{A}^1_{\mathbf{A}}(\{p_4\}) = \mathsf{A}^1_{\mathbf{A}}(\{p_5\}) = \mathsf{A}^1_{\mathbf{A}}(\{p_4, p_5\}) = \{(p_2, \sigma_2), (p_5, \sigma_2)\}.$$

The protagonist's attractor set $\mathsf{A}_{\mathbf{P}}(V')$ is the set of all states from which a visit to V' can be enforced by the protagonist in zero or more steps. $\mathsf{A}_{\mathbf{P}}(V')$ can be computed iteratively via computation of the converging sequence

$$\mathsf{A}^*_{\mathbf{P}0}(V') \subseteq \mathsf{A}^*_{\mathbf{P}1}(V') \subseteq \ldots,$$

where $\mathsf{A}^*_{\mathbf{P}0}(V') = V'$ and

$$\mathsf{A}^*_{\mathbf{P}i+1}(V') = \mathsf{A}^1_{\mathbf{P}}(\mathsf{A}^*_{\mathbf{P}i}(V')) \cup \mathsf{A}^*_{\mathbf{P}i}(V').$$

The sequence is indeed converging because there are at most $|V_{\mathbf{P}} \cup V_{\mathbf{A}}|$ different sets in the sequence. Intuitively $\mathsf{A}^*_{\mathbf{P}i}(V')$ is the set from which a visit to the set V' can be enforced by the protagonist in at most i steps.

Example 10: Consider the Rabin game shown in Figure 8. The protagonist's attractor set for $V' = \{p_4, p_5\}$ is recursively computed as follows:

$$\mathsf{A}^*_{\mathbf{P}1}(V') = \mathsf{A}^*_{\mathbf{P}0}(V') \cup \mathsf{A}^1_{\mathbf{P}}(\{p_4, p_5\}) =$$
$$\{p_4, p_5, (p_2, \sigma_2), (p_5, \sigma_2)\},$$
$$\mathsf{A}^*_{\mathbf{P}2}(V') = \mathsf{A}^*_{\mathbf{P}1}(V') \cup \mathsf{A}^1_{\mathbf{P}}(\mathsf{A}^*_{\mathbf{P}1}(V')) =$$
$$\{p_2, p_4, p_5, (p_2, \sigma_2), (p_5, \sigma_2)\},$$
$$\mathsf{A}^*_{\mathbf{P}}(V') = \mathsf{A}^*_{\mathbf{P}3}(V') = \mathsf{A}^*_{\mathbf{P}2}(V') \cup \mathsf{A}^1_{\mathbf{P}}(\mathsf{A}^*_{\mathbf{P}2}(V')) =$$
$$\{p_2, p_4, p_5, (p_2, \sigma_2), (p_5, \sigma_2)\}.$$

The adversary's attractor set of V' is computed similarly. This computation converges at the fifth iteration, and the resulting set is $\mathsf{A}^*_{\mathbf{A}}(V') = \{p_0, p_2, p_4, p_5, (p_0, \sigma_1), (p_1, \sigma_1), (p_2, \sigma_2), (p_4, \sigma_1), (p_5, \sigma_2)\}$.

Attractor strategy $\pi_{\mathsf{A}_{\mathbf{P}}(V')}$ for the protagonist's attractor set determines how to ensure a visit to set V' from attractor set $\mathsf{A}_{\mathbf{P}}(V')$. For all $v \in \mathsf{A}^*_{\mathbf{P}i+1}(V') \setminus \mathsf{A}^*_{\mathbf{P}i}(V')$, the attractor strategy is defined as $\pi_{\mathsf{A}_{\mathbf{P}}(V')}(v) = (v, v')$, where v' is an arbitrary $v' \in \mathsf{A}^*_{\mathbf{P}i}(V')$. The adversary's attractor $\mathsf{A}_{\mathbf{A}}(V')$ and attractor strategy $\pi_{\mathsf{A}_{\mathbf{A}}(V')}$ are computed analogously. The protagonist's and adversary's attractors of V' in a game $\mathbf{G}$ are denoted by $\mathsf{A}^{\mathbf{G}}_{\mathbf{P}}(V')$ and $\mathsf{A}^{\mathbf{G}}_{\mathbf{A}}(V')$, respectively.

Let (V, E) denote the graph of a Rabin game $G = (V_{\mathbf{P}}, V_{\mathbf{A}}, E, F_{\mathbf{G}})$, where $V = V_{\mathbf{P}} \cup V_{\mathbf{A}}$. For simplicity, for a set $Q \subseteq V$, we denote $\mathbf{G} \setminus Q$ the graph $(V \setminus Q, E \setminus E')$ (and the corresponding game), where E' is the set of all edges incident with states from Q.

Horn's algorithm is summarized in Algorithm 3. First the protagonist chooses a condition (G_i, B_i) (line 1). As the protagonist needs to avoid B_i, a sub game $\mathbf{G}^0_i$ is defined by removing the adversary's attractor set for B_i. Then, a sub game $\mathbf{G}^j_i$ is defined iteratively by removing winning regions for the adversary (line 7). The iterative process terminates when no winning region is found for the adversary, *i.e.,* $\mathbf{G}^j_i = \mathbf{G}^{j+1}_i$. In this case, either $\mathbf{G}^j_i$ is empty, or it is winning for the protagonist. If $\mathbf{G}^j_i$ is not empty, then the protagonists attractor of $\mathbf{G}^j_i$ in game $\mathbf{G}$ (line 11) is also winning for the protagonist. By removing the winning region for the protagonist (line 14), a new smaller game is defined and the algorithm is run on this game.

Example 11: We illustrate Algorithm 3 on the Rabin game shown in Figure 8. At the first iteration, we consider Rabin pair (G_1, B_1), where $G_1 = \{p_4, p_5\}$ and $B_1 = \{p_1, p_2, p_3\}$. The adversary's attractor $\mathsf{A}^{\mathbf{G}}_{\mathbf{A}}(B_1)$ is $V_{\mathbf{P}} \cup V_{\mathbf{A}}$, therefore, on line 10 of Algorithm 3, the graph $\mathbf{G}^0_1$ is empty. As we do not find any states winning for the protagonist, we continue with the next Rabin pair.

In the second iteration of Algorithm 3, we consider Rabin pair (G_2, B_2), where $G_2 = \{p_1, p_2, p_3\}$ and $B_2 = \{p_4, p_5\}$. We eliminate $A^{\mathbf{G}}_{\mathbf{A}}(B_2)$ from the graph on line 3. The remaining graph is $\mathbf{G}^0_2$. We compute $A^{\mathbf{G}^0_2}_{\mathbf{P}}(G_2)$, and find out that it is equal to $\mathbf{G}^0_2$. This means that $\mathbf{H}^0_2$ is empty, $\mathbf{G}^1_2$ is equal to $\mathbf{G}^0_2$, and $\mathbf{G}^0_2$ is guaranteed to be a part of the protagonist's winning region. $A^{\mathbf{G}}_{\mathbf{A}}(B_2)$ and $\mathbf{G}^0_2$ are shown in Figure 9. The protagonist's attractor of $\mathbf{G}^0_2$ in game $\mathbf{G}$ is

$$W_{\mathbf{P}} = A^{\mathbf{G}}_{\mathbf{P}}(\mathbf{G}^0_2) = \{p_1, p_3, p_4, (p_1, \sigma_2), (p_3, \sigma_1), (p_4, \sigma_2)\},$$

and the corresponding winning strategy for the protagonist is (lines 11 and 12)

Algorithm 3 RABINGAME $(G = (V_{\mathbf{P}}, V_{\mathbf{A}}, E, F_{\mathbf{G}}))$: Winning region $W_{\mathbf{P}} \subseteq (V_{\mathbf{P}} \cup V_{\mathbf{A}})$ and winning strategy $\pi_{\mathbf{P}}$ for the protagonist, winning region $W_{\mathbf{A}} \subseteq (V_{\mathbf{P}} \cup V_{\mathbf{A}})$ for the adversary

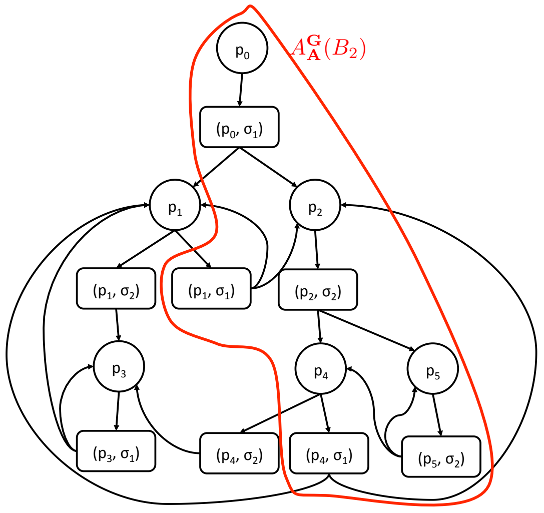
```
1:  for all $(G_i, B_i) \in F_{\mathbf{G}}$ do
2:    $j = 0$
3:    $\mathbf{G}_i^j = \mathbf{G} \setminus \mathsf{A}_{\mathbf{A}}^{\mathbf{G}}(B_i)$   {remove all states in $\mathsf{A}_{\mathbf{A}}^{\mathbf{G}}(B_i)$ and transitions adjacent to them from $\mathbf{G}$}
4:    repeat
5:      $\mathbf{H}_i^j = \mathbf{G}_i^j \setminus \mathsf{A}_{\mathbf{P}}^{\mathbf{G}_i^j}(G_i)$   {note that $(G_i, B_i)$ is not present in $\mathbf{H}_i^j$ any more}
6:      $(W'_{\mathbf{P}}, \pi'_{\mathbf{P}}, W'_{\mathbf{A}})$ = RABINGAME($\mathbf{H}_i^j$)  {recursive call}
7:      $\mathbf{G}_i^{j+1} = \mathbf{G}_i^j \setminus \mathsf{A}_{\mathbf{A}}^{\mathbf{G}_i^j}(W'_{\mathbf{A}})$
8:      $j$++
9:    until $\mathbf{G}_i^j = \mathbf{G}_i^{j+1}$   {$\mathbf{G}_i^j$ is guaranteed to be winning for the protagonist}

10:   if $\mathbf{G}_i^j \neq \emptyset$ then
11:     $W_{\mathbf{P}} = W_{\mathbf{P}} \cup \mathsf{A}_{\mathbf{P}}^{\mathbf{G}}(\mathbf{G}_i^j)$  {The protagonist's attractor of $\mathbf{G}_i^j$ in $\mathbf{G}$ is winning}
12:     $\pi_{\mathbf{P}} = \pi_{\mathbf{P}} \cup \pi'_{\mathbf{P}} \cup \pi''_{\mathbf{P}}$,   {$\pi'_{\mathbf{P}}$ is the protagonist's attractor strategy computed in line 6}
13:          {$\pi''_{\mathbf{P}}$ is the protagonist's attractor strategy for $\mathsf{A}_{\mathbf{P}}^{\mathbf{G}}(\mathbf{G}_i^j)$}
14:     $\mathbf{G}^s = \mathbf{G} \setminus W_{\mathbf{P}}$
15:     $(W_{\mathbf{P}}^s, \pi_{\mathbf{P}}^s, W_{\mathbf{A}}^s)$ = RABINGAME($\mathbf{G}^s$)   {run the algorithm on a smaller graph;
            consider all pairs in the acceptance condition over again}
16:     $W_{\mathbf{P}} = W_{\mathbf{P}} \cup W_{\mathbf{P}}^s$
17:     $\pi_{\mathbf{P}} = \pi_{\mathbf{P}} \cup \pi_{\mathbf{P}}^s$
18:     BREAK   {break the whole for-cycle 1-18}
19:   end if
20: end for
21: $W_{\mathbf{A}} = \mathbf{G} \setminus W_{\mathbf{P}}$
```

$$\pi_{\mathbf{P}}(p_1) = (p_1, (p_1, \sigma_2)),$$
$$\pi_{\mathbf{P}}(p_3) = (p_3, (p_3, \sigma_1)),$$
$$\pi_{\mathbf{P}}(p_4) = (p_4, (p_4, \sigma_2)).$$

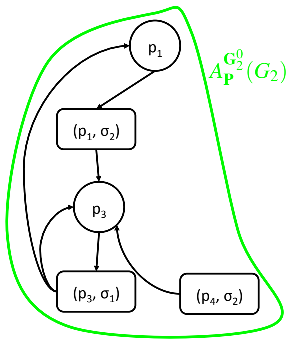
As we find a winning region for the protagonist, we rerun the algorithm for a smaller game (line 15) as illustrated in Figure 10. Note that the algorithm is run from the beginning on the subgame and all Rabin acceptance pairs are considered again.

At the first iteration of Algorithm 3 on the subgame $\mathbf{G}^s$ shown in Figure 10, we consider Rabin pair (G_1^s, B_1^s), where $G_1^s = \{p_5\}$ and $B_1^s = \{p_2\}$. The adversary's attractor of B_1^s is $\{p_0, p_2, (p_0, \sigma_1), (p_1, \sigma_1), (p_4, \sigma_1)\}$, and the protagonist's attractor of G_1^s on $\mathbf{G}_1^0 = \mathbf{G}^s \setminus A_{\mathbf{A}}^{\mathbf{G}^s}(B_1)$ is $\mathbf{G}_1^0$. $\mathbf{H}_1^0$ is empty, and the protagonists wins everywhere in $\mathbf{G}_1^0$ and its attractor in $\mathbf{G}^s$. The attractor of $\mathbf{G}_1^0$ in $\mathbf{G}^s$ covers $\mathbf{G}^s$. Therefore, we find that the protagonist wins everywhere in $\mathbf{G}^s$ with the following strategy:

$$\pi_{\mathbf{P}}^s(p_0) = (p_0, (p_0, \sigma_1)),$$
$$\pi_{\mathbf{P}}^s(p_2) = (p_2, (p_2, \sigma_2)),$$



(a) $A_{\mathbf{A}}^{\mathbf{G}}(B_2)$



(b) Sub-game G_2^0

Fig. 9. Adversary's attractor of B_2 in game $\mathbf{G}$ is shown with a red frame in (a). The sub-game $\mathbf{G}_2^0$ obtained by removing $A_{\mathbf{A}}^{\mathbf{G}}(B_2)$ from $\mathbf{G}$ is shown in (b). The protagonist's attractor of G_2 in game $\mathbf{G}_2^0$ covers $\mathbf{G}_2^0$.

$$\pi_{\mathbf{P}}^s(p_5) = (p_5, (p_5, \sigma_2)).$$

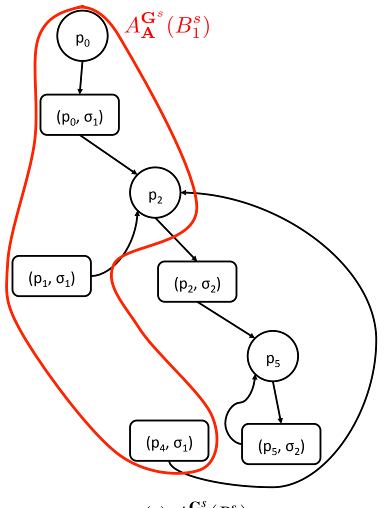
As $W_{\mathbf{P}}^s$ covers $\mathbf{G}^s$, the algorithm (recursive call on the subgame $\mathbf{G}^s$) terminates with $W_{\mathbf{P}}^s$ and strategy $\pi_{\mathbf{P}}^s$. Finally, the winning region for the protagonist $W_{\mathbf{P}}$ on the initial game $\mathbf{G}$ covers $V_{\mathbf{P}} \cup V_{\mathbf{A}}$, and the protagonist wins everywhere in $\mathbf{G}$ with the strategy $\pi_{\mathbf{P}}$ computed in line 17.

Complexity The complexity of Algorithm 3 is $O(|V|^{2n} n!)$. Intuitively, the first part ($O(|V|^{2n})$ comes from the two recursions and the second part ($n!$) comes from the protagonist's ability to change the condition. For a Rabin game of a Rabin automaton, the complexity of the algorithm is $O((|S_P| + |S_P||\Sigma|)^{2n} n!)$, since $V = V_{\mathbf{P}} \cup V_{\mathbf{A}}$, $V_{\mathbf{P}} = S_P$, and $V_{\mathbf{A}} = S_P \times \Sigma$.

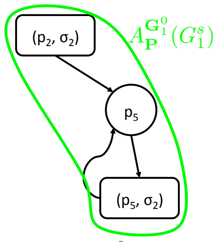
Step 5: Mapping the Rabin Game Solution to a Control Strategy

In order to complete the solution to Problem 1, we transform a solution to a Rabin game $\mathbf{G} = (V_{\mathbf{P}}, V_{\mathbf{A}}, E, F_{\mathbf{G}})$ of the product automaton $P = T \otimes R$ into a control strategy (X_T^{ϕ}, Ω) for T. The solution to the Rabin game is given as a winning region $W_{\mathbf{P}} \subseteq V_{\mathbf{P}}$ and a winning strategy $\pi_{\mathbf{P}} : W_{\mathbf{P}} \to E$.

We first transform the solution into a memoryless strategy for the product P, and present the solution to Problem 2.



(a) $A_{\mathbf{A}}^{\mathbf{G}^s}(B_1^s)$



(b) $A_{\mathbf{P}}^{\mathbf{G}_1^0}(G_1^s)$

Fig. 10. Adversary's attractor set of B_1^s in game $\mathbf{G}^s$ is shown with a red frame in (a). The sub-game $\mathbf{G}_1^0$ is shown in (b). The protagonist's attractor of G_1 in game $\mathbf{G}_1^0$ covers $\mathbf{G}_1^0$.

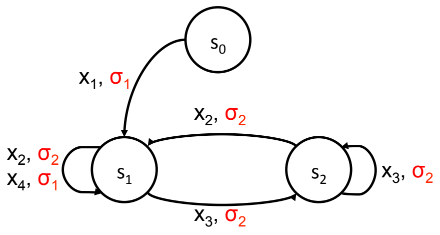


Fig. 11. The control automaton from Example 12. The initial state is s_0 . The arrows between states are labeled with the states of the transition system depicting the memory update function. The corresponding control actions are shown in red.

Clearly, the winning region for P is $W_P = W_{\mathbf{P}}$. The initial winning region is the subset of initial states that belong to W_P, *i.e.*, $W_{P0} = W_P \cap S_{P0}$. The strategy π_P is obtained as follows. For all $v \in W_{\mathbf{P}}$, $\pi_P(v) = \sigma$, such that $\pi_{\mathbf{P}}(v) = (v, v')$, and $v' = (v, \sigma)$.

The remaining task is to adapt (W_{P0}, π_P) as a control strategy (X_T^ϕ, Ω) for T. Although the control function π_P was memoryless, Ω is history dependent and takes the form of a feedback control automaton:

Definition 19: Given a product automaton $P = T \otimes R$, where $T = (X, \Sigma, \delta, O, o)$ and $R = (S, S_0, O, \delta_R, F)$, a winning region W_P for P, and a control strategy (W_{P0}, π_P) for P, a feedback control automaton $C = (S_C, S_{C0}, X, \tau, \Sigma, \pi)$ is defined as

- $S_C = S$ is the set of states,
- $S_{C0} = S_0$ is the set of initial states,
- X is the set of inputs (the set of states of T),
- $\tau : S_C \times X \to S_C$ is the memory update function defined as:

$$\tau(s,x) = \delta_R(s, o(x)) \text{ if } (x,s) \in W_P, \tau(s,x) = \perp \text{ otherwise}$$

- Σ is the set of outputs (the set of inputs of T),
- $\pi : S_C \times X \to \Sigma$ is the output function:

$$\pi(s,x) = \pi_P((x,s)) \text{ if } (x,s) \in W_P, \pi(s,x) = \perp \text{ otherwise.}$$

The set of initial states X_T^ϕ of T is given by $\alpha(W_{P0})$, where $\alpha : S_P \to X$ is the projection from states of P to X. The control function Ω is given by C as follows: for a sequence $x_1 \ldots x_n$, $x_1 \in X_T^\phi$, we have $\Omega(x_1 \ldots x_n) = \sigma$, where $\sigma = \pi(s_n, x_n)$, $s_{i+1} = \tau(s_i, x_i)$, and $x_{i+1} \in \delta(x_i, \pi(s_i, x_i))$, for all $i \in \{1, \ldots, n-1\}$. It is easy to see that the product automaton of T and C will have the same states as P but contains only transitions of P closed under π_P. Then, all trajectories in $T(X_T^\phi, \Omega)$ satisfy ϕ and therefore (X_T^ϕ, Ω) is a solution to Problem 1. Note that if $p = (x,s) \notin W_{\mathbf{P}}$, then the adversary wins all the plays starting from p regardless of the protagonists choices, which implies that there is always a run starting from the product automaton state $(x,s) \in S_P$ that does not satisfy the Rabin acceptance condition F_P regardless of the applied control function. Therefore, Algorithm 2 finds the largest controlled satisfying region.

Example 12: We transform the winning region $W_{\mathbf{P}}$ and the winning strategy $\pi_{\mathbf{P}}$ found in Example 11 into a control strategy (X_T^Φ, Ω) for the transition system T and formula Φ from Example 6. The memoryless control strategy (W_{P0}, π_P) for the product P (Figure 7) is defined as $W_{P0} = \{p_0\}$, $\pi_P(p_0) = \sigma_1, \pi_P(p_1) = \sigma_2, \pi_P(p_2) = \sigma_2, \pi_P(p_3) = \sigma_1, \pi_P(p_4) = \sigma_2$, and $\pi_P(p_5) = \sigma_2$.

The set of initial states is $X_T^\Phi = \{x_1\}$, and the feedback control automaton $C = (S_C, S_{C0}, X, \tau, \Sigma, \pi)$, that defines the history dependent control function Ω, is constructed as in Definition 19. The control automaton is shown in Figure 11 and is formally defined as:

$S_C = \{s_0, s_1, s_2\}$,
$S_{C0} = \{s_0\}$,
$X = \{x_1, x_2, x_3, x_4\}$,
$\tau(s_0, x_1) = s_1,\ \tau(s_1, x_2) = s_1,\ \tau(s_1, x_3) = s_2,\ \tau(s_1, x_4) = s_1,\ \tau(s_2, x_2) = s_1,\ \tau(s_2, x_3) = s_2$,
$\Sigma = \{\sigma_1, \sigma_2\}$,
$\pi(s_0, x_1) = \sigma_1,\ \pi(s_1, x_2) = \sigma_2,\ \pi(s_1, x_3) = \sigma_2,\ \pi(s_1, x_4) = \sigma_1,\ \pi(s_2, x_2) = \sigma_2,\ \pi(s_2, x_3) = \sigma_2$.

B. Control of Transition Systems from dLTL Specifications

In this section, we present a slightly more efficient and intuitive solution to Problem 1 for the case when the LTL specification formula can be translated to a deterministic Büchi automaton. The solution follows the main lines of the method presented in Section V-A for arbitrary LTL specifications. Instead of the Rabin automaton, we construct a deterministic Büchi automaton, and take its product with the transition system. In this case, the product is a nondeterministic Büchi automaton. We find a control strategy for the product by solving a Büchi game and then transform it to a strategy for the original transition system. This procedure is summarized in Algorithm 4. The details are presented in the rest of this section.

The first step of Algorithm 4 is to translate the dLTL specification Φ into a deterministic Büchi automaton $B = (S, S_0, O, \delta_B, F)$. The second step is the construction of a product automaton P of the transition system

Algorithm 4 DLTL CONTROL(T,ϕ) : Control strategy (X_T^ϕ,Ω) such that all trajectories in $T(X_T^\phi,\Omega)$ satisfy ϕ

```
1: Translate $\phi$ to deterministic Büchi automaton $B = (S,S_0,O,\delta_B,F)$
2: Build a product automaton $P = T \otimes B$
3: Solve a Büchi game
4: Map the solution to a control strategy for the original transition system $T$
```

$T = (X,\Sigma,\delta,O,o)$ and B. The product automaton $P = (S_P,S_{P0},\Sigma,\delta_P,F_P)$ is constructed as described in Definition 13 with the exception that the set of accepting states of P is defined as $F_P = X \times F$. The product automaton P is a nondeterministic Büchi automaton if T is nondeterministic, otherwise it is a deterministic Büchi automaton.

Each accepting run $\rho_P = (x_1,s_1)(x_2,s_2)(x_3,s_3)\ldots$ of a product automaton $P = T \otimes B$ can be projected into a trajectory $x_1x_2x_3\ldots$ of T, such that the word $o(x_1)o(x_2)o(x_3)\ldots$ is accepted by B (*i.e.*, satisfies ϕ) and vice versa. Similar to the solution proposed in the previous section, this allows us to reduce Problem 1 to finding a control strategy for P.

Problem 3: Given a controlled Büchi product automaton $P = (S_P,S_{P0},\Sigma,\delta_P,F_P)$, find the largest set of initial states $W_{P0} \subseteq S_{P0}$ for which there exists a control function $\pi_P : S_P \to \Sigma$ such that each run of P under strategy (W_{P0},π_P) satisfies the Büchi accepting condition F_P.

The solution to Problem 3 is summarized in Algorithm 5. The main idea behind the algorithm is to first compute a subset $\overline{F}_P$ of F_P such that a visit to $\overline{F}_P$ can be enforced from $\overline{F}_P$ in a finite number of steps. Then, what remains is to compute the set of all states W_P and a control function π_P such that all runs originating from W_P in closed loop with π_P can reach $\overline{F}_P$ in a one or more steps. By the definition of $\overline{F}_P$, it holds that $\overline{F}_P \subseteq W_P$, and hence these runs satisfy the Büchi condition. To compute $\overline{F}_P$ and π_P, we first define direct and proper attractor sets of a set $S \subseteq S_P$ for a Büchi automaton P:

Definition 20 (Direct attractor): The direct attractor of a set S, denoted by $\mathsf{A}^1(S)$, is defined as the set of all $s \in S_P$ from which there can be enforced a visit to S in one step:

$$\mathsf{A}^1(S) = \{s \in S_P \mid \exists\sigma \in \Sigma, \delta_P(s,\sigma) \subseteq S\}.$$

The direct attractor set induces a strategy $\pi_P^{1,S} : \mathsf{A}^1(S) \to \Sigma$ such that

$$\delta_P(s,\pi_P^1(s)) \subseteq S.$$

Definition 21 (Proper attractor): The proper attractor of a set S, denoted by $\mathsf{A}^+(S)$, is defined as the set of all $s \in S_P$ from which there can be enforced a visit to S in one or more steps.

The proper attractor set $\mathsf{A}^+(S)$ of a set S can be computed iteratively via the converging sequence

$$\mathsf{A}^1(S) \subseteq \mathsf{A}^2(S) \subseteq \ldots,$$

where $\mathsf{A}^1(S)$ is the direct attractor of S, and

$$\mathsf{A}^{i+1}(S) = \mathsf{A}^1(\mathsf{A}^i(S)) \cup \mathsf{A}^i(S).$$

Intuitively, $\mathsf{A}^i(S)$ is the set from which a visit to S in at most i steps can be enforced by choosing proper controls. The *attractor strategy* $\pi_P^{+,S}$ for $\mathsf{A}^+(S)$ is defined from the direct attractor strategies computed through the converging sequence as follows:

$$\pi_P^{+,S} = \pi_P^{1,\mathsf{A}^i(S)}(s), \text{ for all } s \in \mathsf{A}^{i+1}(S) \setminus \mathsf{A}^i(S).$$

Algorithm 5 BÜCHIGAME $(P = (S_P,S_{P0},\Sigma,\delta_P,F_P))$: Winning region $W_P \subseteq S_P$ and winning strategy π_P.

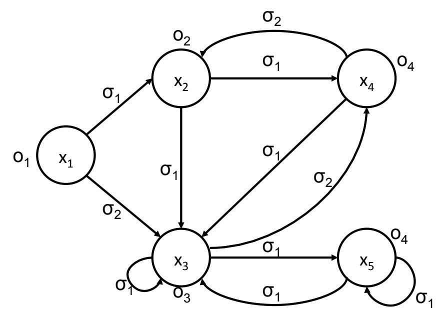
```
1: $\overline{F}_P = \emptyset$
2: $\overline{F}_P^{new} = F_P$
3: **while** $\overline{F}_P \neq \overline{F}_P^{new}$ **do**
4:     $\overline{F}_P = \overline{F}_P^{new}$
5:     $\overline{F}_P^{new} = \mathsf{A}^+(\overline{F}_P) \cap \overline{F}_P$
6: **end while**
7: $W_P = \mathsf{A}^+(\overline{F}_P)$, compute the corresponding attractor strategy $\pi_P$
```

A *recurrent set* of a given set A is the set of states from which infinitely many revisits to A can be enforced. In Algorithm 5, first the recurrent set $\overline{F}_P$ of F_P is computed with an iterative process (lines 3 to 6). Note that we start with $\overline{F}_P = F_P$ and iteratively remove the states from which a revisit to $\overline{F}_P$ can not be guaranteed. This loop terminates after a finite number of iterations since F_P is a finite set. The termination guarantees $\overline{F}_P \subseteq \mathsf{A}^+(\overline{F}_P)$, and hence infinitely many revisits to $\overline{F}_P$ from $\overline{F}_P$ can be enforced. In the final step of the algorithm the proper attractor of $\overline{F}_P$ and the corresponding attractor strategy is computed. As $\overline{F}_P \subseteq \mathsf{A}^+(\overline{F}_P)$, π_P is an attractor strategy that solves the Büchi game for all $s \in W_P$.

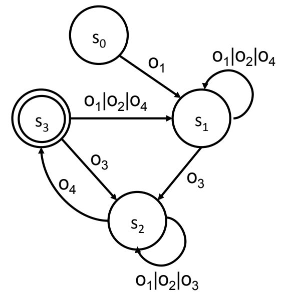
Complexity The time complexity of Algorithm 5 is $O(|S_P|^2|\Sigma|)$.

Remark 1: A Büchi automaton B can be interpreted as a Rabin automaton with a single pair $\{(G_1,B_1)\}$ in its accepting condition, where $G_1 = \overline{F}_P$ and $B_1 = \emptyset$. Consequently, Algorithm 3 for the Rabin game can be used for the Büchi automaton to solve Problem 3. In this particular case, $n = 1$ and the time complexity of Algorithm 3 is $O((|S_P| + |S_P||\Sigma|)^2)$.

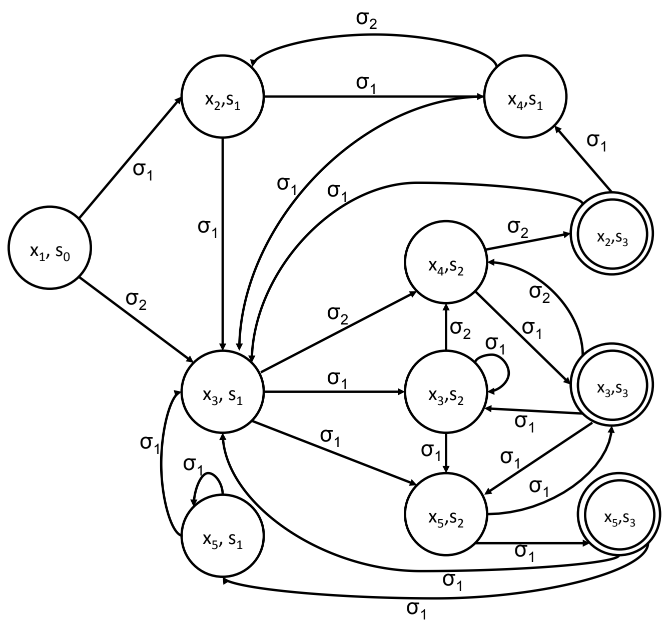
The final step of the dLTL control algorithm is to translate the control strategy (W_{P0},π_P) obtained from Algorithm 5 into a control strategy (X_T^ϕ,Ω) for T, where $W_{P0} = W_P \cap S_{P0}$. As in the solution presented for LTL specifications in the previous section, although the control function π_P is memoryless, Ω is history dependent and takes the form of a feedback control automaton. The control automaton is constructed from P and π_P as in Definition 4, and the control function Ω is defined by the control automaton. Finally, the set of initial states X_T^ϕ of T is given by $\alpha(W_{P0})$, where $\alpha : S_P \to X$ is the projection from states of P to X.



(a) Transition system T



(b) $\phi = o_1 \wedge \Box(\diamond o_3 \wedge \diamond o_4)$



(c) The product of T (a) and B (b)

Fig. 12. Transition system (a), Büchi automaton (b), and their product (c) from Example 13. For the Büchi automaton, s_0 is the initial state and s_3 is the accepting state. For the product automaton, (x_1,s_0) is the initial state, and $\{(x_2,s_3),(x_3,s_3),(x_5,s_3)\}$ is the set of accepting states. The states that are not reachable from the non-blocking initial state (x_1,s_0) are not shown in (c).

Example 13: Consider the nondeterministic transition system T shown in Figure 12(a) and the following LTL formula over its set of observations:

$$\phi = o_1 \wedge \Box(\diamond o_3 \wedge \diamond o_4).$$

We follow Algorithm 4 to find the control strategy (X_T^{Φ},Ω) that solves Problem 1 for the transition system T and formula ϕ.

We first construct a deterministic Büchi automaton B (Figure 12(b)) that accepts the language satisfying the formula. Then, we construct the product of the system and the automaton. The product automaton P, which is shown in Figure 12(c), is a non-deterministic Büchi automaton since T is nondeterministic. Note that the states that are not reachable from non-blocking initial states are removed from P and are not shown in Figure 12(c).

To find a control strategy for P, we follow Algorithm 5. In the first iteration, $\overline{F}_P = \{(x_2,s_3),(x_3,s_3),(x_5,s_3)\}$ (F_P) and we compute the proper attractor of F_P as follows: $\mathsf{A}^1(F_P) = \{(x_4,s_2),(x_5,s_2)\}$, $\mathsf{A}^2(F_P) = \{(x_3,s_1),(x_3,s_2),(x_3,s_3),(x_4,s_2),(x_5,s_2)\}$, $\mathsf{A}^3(F_P) = \{(x_1,s_0),(x_4,s_1),(x_3,s_1),(x_3,s_2),(x_3,s_3),(x_4,s_2),(x_5,s_2)\}$, $\mathsf{A}^4(F_P) = \{(x_2,s_1),(x_2,s_3),(x_1,s_0),(x_4,s_1),(x_3,s_1),(x_3,s_2),(x_3,s_3),(x_4,s_2),(x_5,s_2)\}$.

The sequence converges at iteration 4 and $\mathsf{A}^+(F_P) = \mathsf{A}^4(F_P)$. In the first iteration of the main loop of Algorithm 5, $\overline{F}_P^{new} = \{(x_2,s_3),(x_3,s_3)\}$, and (x_5,s_3) is eliminated. The main loop terminates after the second iteration as

$$\mathsf{A}^+(\overline{F}_P) \cap \overline{F}_P = \overline{F}_P, \text{ where } \overline{F}_P = \{(x_2,s_3),(x_3,s_3)\}.$$

As the last step of Algorithm 5, we compute $W_P = \mathsf{A}^+(\{(x_2,s_3),(x_3,s_3)\})$, and the corresponding attractor strategy as follows: $\mathsf{A}^1(\overline{F}_P) = \{(x_4,s_2)\}$, $\pi_P((x_4,s_2)) = \sigma_1$, $\mathsf{A}^2(\overline{F}_P) = \{(x_3,s_3),(x_3,s_2),(x_3,s_1)\} \cup \mathsf{A}^1(\overline{F}_P)$, $\pi_P((x_3,s_3)) = \sigma_2$, $\pi_P((x_3,s_2)) = \sigma_2$, $\pi_P((x_3,s_1)) = \sigma_2$, $\mathsf{A}^3(\overline{F}_P) = \{(x_1,s_0),(x_4,s_1)\} \cup \mathsf{A}^2(\overline{F}_P)$, $\pi_P((x_1,s_0)) = \sigma_2$, $\pi_P((x_4,s_1)) = \sigma_1$, $\mathsf{A}^4(\overline{F}_P) = \{(x_2,s_1),(x_2,s_3)\} \cup \mathsf{A}^3(\overline{F}_P)$, $\pi_P((x_2,s_1)) = \sigma_1$, $\pi_P((x_2,s_3)) = \sigma_1$, $\mathsf{A}^4(\overline{F}_P) = \mathsf{A}^5(\overline{F}_P) = \mathsf{A}^+(\overline{F}_P)$.

The control strategy (W_{P0},π_P) solves Problem 3 for P, where $W_{P0} = \{(x_1,s_0)\}$ and π_P is as defined above. The final step is the transformation of (W_{P0},π_P) into a control strategy $(X_T^{\Phi},\Omega)$for T. The set of initial states is $X_T^{\Phi} = \{x_1\}$, and the feedback control automaton $C = (S_C,S_{C0},X,\tau,\Sigma,\pi)$ (shown in Figure 13), which defines the history dependent control function Ω, is constructed as in Definition 19, and formally defined as:

$S_C = \{s_0,s_1,s_2,s_3\}$,
$S_{C0} = \{s_0\}$,
$X = \{x_1,x_2,x_3,x_4,x_5\}$,
$\tau(s_0,x_1) = s_1$, $\tau(s_1,x_2) = s_1$, $\tau(s_1,x_3) = s_2$, $\tau(s_1,x_4) = s_1$, $\tau(s_2,x_3) = s_2$, $\tau(s_2,x_4) = s_3$, $\tau(s_3,x_2) = s_1$, $\tau(s_3,x_3) = s_2$
$\Sigma = \{\sigma_1,\sigma_2\}$,
$\pi(s_0,x_1) = \sigma_2$, $\pi(s_1,x_2) = \sigma_1$, $\pi(s_1,x_3) = \sigma_2$, $\pi(s_1,x_4) = \sigma_1$, $\pi(s_2,x_3) = \sigma_2$, $\pi(s_2,x_4) = \sigma_1$, $\pi(s_3,x_2) = \sigma_1$, $\pi(s_3,x_3) = \sigma_2$.

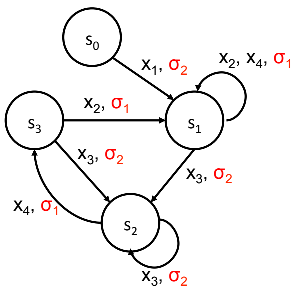


Fig. 13. The control automaton obtained by solving the Büchi game on the product automaton shown in Figure 12(c).

Remark 2: In a Büchi game over a product automaton $P = T \otimes B$, a state (x,s) is added to W_P only if there is a control strategy guaranteeing that all runs ρ_P of P originating from (x,s) satisfy that the projection of ρ_P onto S (Büchi automaton states) is an accepting run of B. The condition is necessary to guarantee that each run of T that originate from x is satisfying. While the product is a non-deterministic Büchi automaton, this condition is stronger than the Büchi acceptance: a word is accepted by a non-deterministic Büchi automaton if there exists an accepting run. In other words, it is not necessary that all runs are accepting. Due to this difference in the notion of the satisfying TS states and non-deterministic Büchi acceptance, an algorithm similar to Algorithm 4 cannot be used for non-deterministic Büchi automaton, which is illustrated in Example 14.

Example 14: Consider the transition system T shown in Figure 14(a) and specification $\phi = \diamond o_2$ over its set of observations. A deterministic Büchi automaton and a non-deterministic Büchi automaton that accept the language satisfying the formula are shown in Figures 14(b) and 14(d), respectively. The corresponding product automata are shown in Figures 14(c) and 14(e). T has a single run $x_2x_2x_2\ldots$ originating from x_2 and it satisfies ϕ_1. Due to the non-determinism of T, there are multiple runs originating from x_1. The run $x_1x_1x_1\ldots$ originating from x_1 produces the word $o_1o_1o_1\ldots$ that violates the formula, and all other runs originating from x_1 satisfy the formula. Therefore, we have $X_T^\phi = \{x_2\}$. We can easily verify this observation by running Algorithm 5 on the product automaton P_1 with $\Sigma = \{\sigma\}$, *i.e.,* a single control input labels all the transitions. The algorithm returns $W_P = \mathsf{A}^+(\overline{F}_P) = \{(x_2,s_0),(x_2,s_1)\}$, hence $W_{P0} = \{(x_2,s_0)\}$ and $X_T^\phi = \{x_2\}$.

Now, consider the product P_2 (Figure 14(e)) of T and the non-deterministic Büchi automaton B_2 accepting the same language as B_1. It is not possible to differentiate (x_1,s_0) and (x_2,s_0) on P_2 via reachability analysis or recurrence set construction, since satisfying and violating runs originate from both (x_1,s_0) and (x_2,s_0).

VI. Temporal Logic Control for Piecewise Affine Systems

Piecewise affine systems (PWA), *i.e.,* systems that evolve along different discrete-time affine dynamics in different

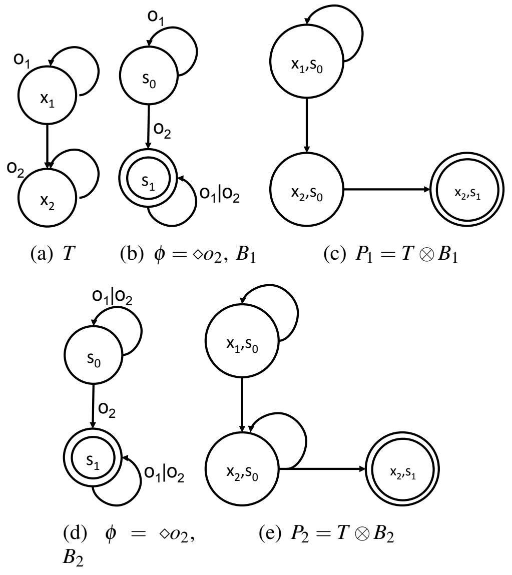


Fig. 14. Transition system T (a), Büchi automata B_1 (b) and B_2 (d), the product of T and B_1 (c), and the product of T and B_2 (e) from Example 14

polytopic regions of the (continuous) state space are widely used as models in many areas. They can approximate non-linear dynamics with arbitrary accuracy and are equivalent with several other classes of systems, including hybrid systems [51]. In addition, there exist efficient techniques for the identification of such models from experimental data, which include Bayesian methods, bounded-error procedures, clustering-based methods, mixed-integer programming, and algebraic geometric methods (see [52], [53] for a review).

In this section, we consider the problem of controlling a discrete-time PWA system from an LTL specification over linear predicates in its state variables. The material presented in this section is based on [54], [38], [13]. Related approaches can be found in [10] and [55], where the existence of equivalent (bisimulation) quotients and control strategies under the assumption of controllability for discrete-time linear systems is characterized. The monotonicity property of a discrete-time piecewise system is exploited in [56] to develop an efficient abstraction algorithm.Algorithmic procedures for controlling continuous-time linear systems are given in [15], [41], where the constructed deterministic (nondeterministic) abstractions are not equivalent to the original system but instead capture only a restricted but controllable subset of its behavior. An alternative abstraction technique based on quantifier elimination for real closed fields and theorem proving has been proposed in [57].

To solve Rabin games, we use the algorithm from [37] (see details in Sec. V) but extended it to deal with stuttering behavior, which leads to additional winning strategies. The concept of stuttering has been established in [19] and related work has focused on determining if a specification is closed under stuttering [58], in which case it can be expressed in the LTL$\setminus\bigcirc$ fragment (LTL without the next operator) [59] and less conservative stutter bisimulation quotients can be constructed [19]. The approach from this section does not require any special structure from either the specification or the quotient. Instead, we characterize individual transitions as stuttering while constructing the abstraction and use this additional information during the solution of the Rabin game, which reduces the conservatism of the overall method but does not restrict the expressivity of the specification. Another related approach is based on characterizing the sets of transient states (instead of an individual transition) and augmenting the transition relation with the corresponding transitions, which was applied to switched systems in [60]. While considering transient sets reduces conservatism, it is computationally expensive to characterize such sets.

A. Discrete-Time Piecewise Affine Systems

Let L be a finite index set and $\mathbf{X}_l$, $l \in L$ be a set of open, full dimensional polytopes [3] in $\mathbb{R}^N$, such that $\mathbf{X}_{l_1} \cap \mathbf{X}_{l_2} = \emptyset$ for all $l_1, l_2 \in L$, where $l_1 \neq l_2$.

Definition 22 (PWA Control System): A discrete-time piecewise affine (PWA) control system $\mathscr{W}$ over $\mathbf{X} = \bigcup_{l \in L} \mathbf{X}_l$ is defined as:

$$\mathscr{W} : x_{k+1} = A_l x_k + B_l u_k + c_l,\ x_k \in \mathbf{X}_l,\ u_k \in \mathbf{U},\ l \in L \quad (24)$$

where, at each time step $k = 0, 1, \ldots$, $x_k \in \mathbb{R}^N$ is the state of the system and u_k is the input restricted to a polytopic set $\mathbf{U} \subset \mathbb{R}^M$. Matrices $A_l \in \mathbb{R}^{N \times N}$, $B_l \in \mathbb{R}^{N \times M}$, $c_l \in \mathbb{R}^N$ are the system parameters for mode $l \in L$.

System $\mathscr{W}$ evolves along different affine dynamics in different regions of the continuous state space $\mathbf{X}$. When $\mathscr{W}$ is in a state $x_k \in \mathbf{X}_l$ for some $l \in L$, we say that the system is in *mode* $l \in L$. Then, the next visited state x_{k+1} is computed according to the affine map of Equation (24) with parameters A_l and c_l, specifying the dynamics of $\mathscr{W}$ in mode l. Starting from initial conditions $x_0 \in \mathbf{X}_{l_0}$ for some $l_0 \in L$ a trajectory of system $\mathscr{W}$ can be obtained by the following (numerical simulation) procedure:

i) Start from initial conditions $x_0 \in \mathbf{X}_{l_0}$, where the system is in mode l_0.
ii) Select an input $u \in \mathbf{U}$ from the allowed input set.
iii) Apply the affine map of Definition 22 to compute the next state $x_1 = A x_0 + B_{l_0} u + c$.
iv) Find the mode $l_1 \in L$ of $\mathscr{W}$ such that $x_1 \in \mathbf{X}_{l_1}$.
v) Repeat this procedure iteratively for each subsequent step.

The operating regions $\mathbf{X}_l, l \in L$ from Definition 22 of a PWA system are also considered as regions of interests of system $\mathscr{W}$. As we are only interested in trajectories of system $\mathscr{W}$ evolving in $\mathbf{X}$, we define an additional mode Out with trivial dynamics $x_{k+1} = x_k$ and region $\mathbf{X}_{\text{Out}} = \mathbb{R}^N \setminus \mathbf{X}$. Informally, a trajectory $w_\mathbf{X} = w_\mathbf{X}(1) w_\mathbf{X}(2) \ldots$ produces a word $w_L = w_L(1) w_L(2) \ldots$ such that $w_L(i)$ is the index of the region visited by state $w_\mathbf{X}(i)$, *i.e.,* $w_\mathbf{X}(i) \in \mathbf{X}_{w_L(i)}$, and $w_L(i)$ is Out if $w_\mathbf{X}(i) \notin \mathbf{X}$. For example, trajectory $x_0 x_1 x_2 \ldots$ satisfying $x_0, x_1 \in \mathbf{X}_{l_1}$ and $x_2 \in \mathbf{X}_{l_2}$ for some $l_1, l_2 \in L$ produces word $l_1 l_1 l_2 \ldots$. After a short example, we formalize the semantics of PWA trajectories and their satisfaction of LTL formulas through an embedding into a transition system, which generalizes the one already introduced in Section II-B.

[3]In this paper, we assume polytopes are open and full dimensional, unless specifically mentioned otherwise.

Definition 23 (Embedding Transition System for $\mathscr{W}$): The embedding for PWA control system $\mathscr{W}$ (Definition 22) is a transition system (Definition 1) $T_\mathscr{W} = (X_\mathscr{W}, \Sigma_\mathscr{W}, \delta_\mathscr{W}, O_\mathscr{W}, o_\mathscr{W})$ with:

- $X_\mathscr{W} = \mathbb{R}^n$,
- $\Sigma_\mathscr{W} = \mathbf{U}$,
- $\delta_\mathscr{W}(x, u) = A_l x + B_l u + c_l$, if $x \in \mathbf{X}_l$,
- $O_\mathscr{W} = L \cup \{\text{Out}\}$,
- $o_\mathscr{W}(x) = l$ if and only if there exists $l \in L$ such that $x \in \mathbf{X}_l$ and $o_\mathscr{W}(x) = \text{Out}$ otherwise.

The embedding transition system from Definition 23 has an infinite number of states and inputs and is non-blocking. Only infinite words are produced by the embedding transition system and, therefore, LTL formulas over $L \cup \{\text{Out}\}$ can be interpreted over such words, leading to the following definition:

Definition 24 (LTL satisfaction for $\mathscr{W}$): Trajectories of a PWA system $\mathscr{W}$ (Definition 22) originating in a polytope $\mathbf{X}_0 \subseteq \mathbf{X}$ satisfy formula ϕ if and only if $T_\mathscr{W}(\mathbf{X}_0)$ satisfies ϕ.

We made some simplifying assumptions in our definition of the PWA system $\mathscr{W}$ and its embedding (Definitions 22 and 23), which is inspired from [9], [10], [11], [18] (see also [12], [13], [14], [61], [62], [15], [16], [17]). First, we defined the system on a set of open full dimensional polytopes, thus ignoring states where the dynamics are ambiguous (states on the boundaries between regions). This is enough for practical purposes, since only sets of measure zero are disregarded and it is unreasonable to assume that equality constraints can be detected in real-world applications. Trajectories starting and remaining in such sets are therefore of no interest. Trajectories starting in the interior of full-dimensional polytopes also cannot "vanish" in such zero-measure sets unless the dynamics of the system satisfy some special conditions, which are easy to derive but omitted. Furthermore, if such sets are of interest, the results presented throughout the book can be extended to more general case where the state space is partitioned into poytopes with some of the facets removed. In particular, facets are simply lower dimensional polytopes and the results can be extended by induction. Second, the semantics is defined over the polytopes $\mathbf{X}_l$, which are given *a priori*. However, arbitrary linear inequalities can be accommodated by including additional polytopes (as long as the region $l \in L$ visited at each step can be observed), in which case the system will have the same dynamics in several modes.

In this section, we consider the following problem:

Problem 4 (PWA Control): Given a PWA control system $\mathscr{W}$ (Definition 22) and an LTL formula ϕ over $L \cup \{\text{Out}\}$, find a control strategy such that all trajectories of the closed loop system satisfy ϕ.

Using Definition 23, Problem 4 becomes an LTL control problem, where we seek a feedback control strategy $(\mathbf{X}_0, \Omega)$ for the infinite, deterministic embedding transition system $T_\mathscr{W}$. We assume that the state of the system cannot be measured precisely or the applied inputs are corrupted by noise and, therefore, seek control strategies that are robust both with respect to measured state and applied input. As a result, the set of initial states $\mathbf{X}_0$ and control function Ω from the feedback control strategy $(\mathbf{X}_0, \Omega)$ (see Definition 11) are defined in terms of the initial regions of $\mathscr{W}$ and no state refinement is performed. In Section V, we discussed the problem of controlling a finite, possibly nondeterministic transition system from LTL specifications. To apply the methods presented there to Problem 4, we develop an approach based on the construction of a finite abstraction for $T_{\mathscr{W}}$, which we refer to as the control transition system T_c such that a control strategy generated for T_c using the algorithms from Section V can be adapted for $T_{\mathscr{W}}$.

More specifically, we construct T_c through a two step process. In the first step, by using the state equivalence relation induced by the polytopes from the definition of the PWA system, we construct a quotient $T_{\mathscr{W}}/_\sim$, which has finitely many states but an infinite set of inputs. Once the quotient $T_{\mathscr{W}}/_\sim$ is constructed, we then define an equivalence relation in the control space, which leads to the construction of the finite control transition system T_c, which is suitable for the methods from Section V. The solution to Problem 4 is obtained by implementing the control strategy for T_c as a feedback control automaton for the initial PWA system that reads the index of the region visited at each step and supplies the next input. As it will become clear later, our approach is robust in the sense that the closed loop system is guaranteed to satisfy the specification, even when state measurements and applied inputs are perturbed.

The remainder of this section is organized as follows. In Section VI-B we define the control transition system T_c and outline an algorithm for its computation. In Section VI-C we show how the methods from Section V are applied to T_c to formulate a solution to Problem 4. In Section VI-D we discuss a strategy for reducing the conservatism of the overall method by characterizing the *stuttering behavior* (self transitions at a state of T_c that can be taken infinitely in T_c but do not correspond to real trajectories of $T_{\mathscr{W}}$) inherent in the construction of the control transition system. This approach is related to the well known Zeno behavior.

B. Control Abstraction

In this section, we define the finite control transition system $T_c = (X_c, \Sigma_c, \delta_c, O_c, o_c)$ for the (infinite) embedding $T_{\mathscr{W}} = (X_{\mathscr{W}}, \Sigma_{\mathscr{W}}, \delta_{\mathscr{W}}, O_{\mathscr{W}}, o_{\mathscr{W}})$ (Definition 23) and present an algorithm for its computation.

1) Definition: The observation map $o_{\mathscr{W}}$ of $T_{\mathscr{W}}$ induces an observational equivalence relation $\sim$ over the set of states $X_{\mathscr{W}}$. The quotient transition system $T_{\mathscr{W}}/_\sim = (X_{\mathscr{W}}/_\sim, \Sigma_{\mathscr{W}}, \delta_{\mathscr{W},\sim}, O_{\mathscr{W}}, o_{\mathscr{W},\sim})$ induced by $\sim$ has an infinite set of inputs $\Sigma_{\mathscr{W}} = \mathbf{U}$, which is preserved from $T_{\mathscr{W}}$. The set of transitions of $T_{\mathscr{W}}/_\sim$ is defined as $l' \in \delta_{\mathscr{W},\sim}(l,u)$ if and only if there exist $u \in \mathbf{U}, x \in \mathbf{X}_l$ and $x' \in \mathbf{X}'$ such that $x' = \delta_{\mathscr{W}}(x,u)$. Note that in general $T_{\mathscr{W}}/_\sim$ is nondeterministic, even though $T_{\mathscr{W}}$ is deterministic. Indeed, for a state of the quotient $l \in X_{\mathscr{W}}/_\sim$ it is possible that different states $x, x' \in \mathbf{X}_l$ have transitions in $T_{\mathscr{W}}$ to states from different equivalence classes under the same input. The set of observations $O_{\mathscr{W}} = L$ of $T_{\mathscr{W}}/_\sim$ is preserved from $T_{\mathscr{W}}$ and the observation map $o_{\mathscr{W},\sim}$ is identity.

The transition map $\delta_{\mathscr{W},\sim}$ can be related to the transitions of $T_{\mathscr{W}}$ by using the *Post* operator:

$$\delta_{\mathscr{W},\sim}(l,u) = \{l' \in X_{\mathscr{W}}/_\sim \mid Post(\mathbf{X}_l, \{u\}) \cap \mathbf{X}_{l'} \neq \emptyset\}, \quad (25)$$

for all $l \in X_{\mathscr{W}}/_\sim$ and $u \in \Sigma_{\mathscr{W}}$. For each state $l \in X_{\mathscr{W}}/_\sim$, we define an equivalence relation $\approx_l$ over the set of inputs $\Sigma_{\mathscr{W}}$ as

$$(u_1, u_2) \in \approx_l \text{ iff } \delta_{\mathscr{W},\sim}(l,u_1) = \delta_{\mathscr{W},\sim}(l,u_2). \quad (26)$$

In other words, inputs u_1 and u_2 are equivalent at state l if they produce the same set of transitions in $T_{\mathscr{W}}/_\sim$. Let U_l^C, $l \in L$, $C \in 2^{X_{\mathscr{W}}/_\sim}$ denote the equivalence classes of $\Sigma_{\mathscr{W}}$ in the partition induced by the equivalence relation $\approx_l$:

$$U_l^C = \{u \in \Sigma_{\mathscr{W}} \mid \delta_{\mathscr{W},\sim}(l,u) = C\} \quad (27)$$

Let $c(U_l^C)$ be an input in U_l^C such that

$$\forall u \in \Sigma_{\mathscr{W}}, d(c(U_l^C),u) < \varepsilon \Rightarrow u \in U_l^C, \quad (28)$$

where $d(u,u')$ denotes the distance between inputs $u, u' \in \Sigma_{\mathscr{W}}$ and ε is a predefined parameter specifying the robustness of the control strategy. As it will become clear in Section VI-B.2, $d(u,u')$ is the Euclidian distance in $\mathbb{R}^M$ and $c(U_l^C)$ can be computed as the center of a sphere inscribed in U_l^C.

Initially, the states of T_c are the observations of $T_{\mathscr{W}}$ (*i.e.*, $X_c = L$). The set of inputs available at a state $l \in L$ is $\Sigma_c^l = \{c(U_l^C) \mid C \in 2^{X_{\mathscr{W}}/_\sim}\}$ and the transition map is $\delta_c(l, c(U_l^C)) = C$. In general, it is possible that at a given state $l, \Sigma_c^l = \emptyset$, in which case state l is blocking. As it will become clear in Section VI-B.2, such states are removed from the system in a recursive procedure together with their incoming transitions and therefore $X_c \subseteq L$. The set of X_c observations and observation map of T_c are preserved from $T_{\mathscr{W}}/_\sim$, which completes the construction of the control transition system.

Following from the construction described so far, the control transition system T_c is a finite transition system and a control strategy for it can be generated using the methods presented in Section V. In Section VI-C, we will show that a control strategy for T_c can be adapted as a robust control strategy (with respect to knowledge of exact state and applied input) for the infinite $T_{\mathscr{W}}$. This will allow us to use T_c as part of our solution to Problem 4.

2) Computation: Initially, the states of the control transition system T_c are simply the labels L of the polytopes from the definition of the PWA system (Definition 22). To complete its construction, we need to be able to compute the set of inputs Σ_c^l available at each state $l \in X_c$ and the transition map δ_c, while eliminating the states that are unreachable in order to guarantee that T_c remains non-blocking.

Given a polytope $\mathbf{X}_l$ from the definition of the PWA system, let

$$\Sigma^l = \{u \in \Sigma_{\mathscr{W}} \mid Post_{T_{\mathscr{W}}}(\mathbf{X}_l, u) \subseteq \mathbf{X}\} \quad (29)$$

be the set of all inputs guaranteeing that all states from $\mathbf{X}_l$ transit inside $\mathbf{X}$ (*i.e.*, Σ^l is the set of all inputs allowed at l). In other words, regardless which $u \in \Sigma^l$ and $x \in \mathbf{X}_l$ are selected, x will transit inside $\mathbf{X}$ under u in $T_{\mathscr{W}}$. Then, in order to guarantee that $\mathbf{X}$ is an invariant for all trajectories of the system (an assumption that we made in the formulation of Problem 4) it is sufficient to restrict the set of inputs Σ_c^l available at each state $l \in X_c$ to $\Sigma_c^l \subseteq \Sigma^l$.

Proposition 1: Let $\mathbf{X} = \{x \in \mathbb{R}^N \mid Hx < K\}$ be the H-representation of the polytope $\mathbf{X}$ from the definition of the PWA system (Definition 22). Then, Σ^l is a polytope with the following H-representation:

$$\Sigma^l = \{u \in \mathbf{U} \mid \forall v \in V(\mathbf{X}_l), HB_l u < K - H(A_l v + c_l)\}, \quad (30)$$

where $V(\mathbf{X}_l)$ denotes the set of vertices of $\mathbf{X}_l$.

The set of states reachable from state l in $T_{\mathscr{W}}/_{\sim}$ under the allowed inputs is

$$Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma^l) = \{l' \in X_{\mathscr{W}}/_{\sim} \mid Post_{T_{\mathscr{W}}}(\mathbf{X}_l, \Sigma^l) \cap \mathbf{X}_{l'} \neq \emptyset\} \quad (31)$$

and can be computed using polyhedral operations, since

$$Post_{T_{\mathscr{W}}}(\mathbf{X}_l, \Sigma^l) = A_l \mathbf{X}_l + B_l \Sigma^l + c_l. \quad (32)$$

Given a polytope $\mathbf{X}_l$ from the definition of the PWA system (Definition 22) and an arbitrary polytope $\mathbf{X}'$, let

$$U^{\mathbf{X}_l \to \mathbf{X}'} = \{u \in \Sigma_{\mathscr{W}} \mid Post_{T_{\mathscr{W}}}(\mathbf{X}_l, u) \cap \mathbf{X}' \neq \emptyset\} \quad (33)$$

denote the set of all inputs under which $T_{\mathscr{W}}$ can make a transition from a state in $\mathbf{X}_l$ to a state inside $\mathbf{X}'$. Equivalently, applying any input $u \in \mathbf{U}, u \notin U^{\mathbf{X}_l \to \mathbf{X}'}$ guarantees that $T_{\mathscr{W}}$ will not make a transition inside $\mathbf{X}'$, from any state in $\mathbf{X}_l$. The following proposition states that $U^{\mathbf{X}_l \to \mathbf{X}'}$ is a polyhedral set that can be computed from the V- (vertex) and H- (hyperplane) representations of $\mathbf{X}_l$ and $\mathbf{X}'$:

Proposition 2: Let H and K be the matrices in the H-representation of the following polytope:

$$\{\hat{x} \in \mathbb{R}^N \mid \exists x \in \mathbf{X}_l, A_l x + \hat{x} + c_l \in \mathbf{X}'\} \quad (34)$$

Then $U^{\mathbf{X}_l \to \mathbf{X}'}$ is a polytope with the following H-representation:

$$U^{\mathbf{X}_l \to \mathbf{X}'} = \{u \in \mathbf{U} \mid HB_l u < K\} \quad (35)$$

Proposition 3: Given a state $l \in X_c$ and a set of states $C \in 2^{X_c}$, the set U_l^C from Equation (27) can be computed as follows:

$$U_l^C = \bigcap_{l' \in C} U^{\mathbf{X}_l \to \mathbf{X}_{l'}} \setminus \bigcup_{l'' \notin C} U^{\mathbf{X}_l \to \mathbf{X}_{l''}} \quad (36)$$

We can guarantee that if a state l' is not reachable from state l in $T_{\mathscr{W}}/_{\sim}$ (*i.e.*, $l' \notin Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma^l)$) then $U^{\mathbf{X}_l \to \mathbf{X}_{l'}} = \emptyset$ and therefore, $U_l^C = \emptyset$ if $C \nsubseteq Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma^l)$ and otherwise the computation in Equation (36) reduces to

$$U_l^C = \bigcap_{l' \in C} U^{\mathbf{X}_l \to \mathbf{X}_{l'}} \setminus \bigcup_{l'' \in Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma^l) \setminus C} U^{\mathbf{X}_l \to \mathbf{X}_{l''}} \quad (37)$$

A non-empty input region U_l^C is in general nonconvex but can always be represented as a finite union of open polytopes (see Equation (37)). In order to guarantee the robustness of

Algorithm 6 T_c = CONTROL-TS($\mathscr{W}$,ε): Construct control transition system T_c

```
 1: $X_c := L$
 2: for each $l \in X_c$ do
 3:     $\Sigma_c^l := \Sigma^l$ [Equation (29))]
 4:     compute $Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma_c^l)$ [Equation (31))]
 5:     for each $C \subseteq Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma_c^l)$ do
 6:         compute $U_l^C$ [Equation (37))]
 7:         if $r(U_l^C) > \varepsilon$ then
 8:             include input $c(U_l^C)$ in $\Sigma_c^l$
 9:             include transition $\delta_c(l, c(U_l^C)) = C$
10:         end if
11:     end for
12:     if $\Sigma_c^l = \emptyset$ then
13:         recursively make state $l$ unreachable and set $X_c := X_c \setminus l$
14:     end if
15: end for
16: $\Sigma_c = \bigcup_{l \in X_c} \Sigma_c^l$
17: return $T_c$
```

the control strategy (as described in Section VI-B.1) we only include input sets that are "large enough" (*i.e.*, $r(U_l^C) > \varepsilon$, where ε is a predefined robustness parameter and $r()$ is the radius of the Chebyshev ball. Note that in general this approach might be conservative, since a sphere inscribed in a union of polytopes from U_l^C might have a larger radius. Following from the results presented in this section, the control transition system T_c can be computed using polyhedral operations only (the computation is summarized in Algorithm 6).

Remark 3: It is possible to reduce the size of T_c after it is initially constructed without sacrificing solutions. More "nondeterminism" available at a state does not result in more winning strategies for Algorithm 6, while at the same time unnecessarily increases the complexity of the method. Formally, let $u_1 = c(U_l^{C_1})$ and $u_2 = c(U_l^{C_2})$ where $C_1, C_2 \in 2_c^X, C_1 \subseteq C_2$ be inputs of T_c available at state $l \in X_c$ (*i.e.*, $\{u_1, u_2\} \subseteq \Sigma_c^l$). If input u_2 is used in a control strategy, then the specification is satisfied regardless of which state $l' \in C_2$ is visited in the next step. Clearly, the same holds for input u_1 since C_1 is a subset of C_2 but keeping both inputs is unnecessary. Therefore, at each state $l \in X_c$ we set $\Sigma_c^{ls} = \Sigma_c^{ls} \setminus u_2$ if $u_1, u_2 \in \Sigma_c^{ls}$ or $\Sigma_c^{lu} = \Sigma_c^{lu} \setminus u_2$ if $u_1, u_2 \in \Sigma_c^{lu}$ when the property described above holds.

C. LTL Control of PWA Systems

In Section VI-B we defined the control transition system T_c as a finite abstraction of the infinite $T_{\mathscr{W}}$ and showed that it can be computed using polyhedral operations. In Section V-A, we presented an approach for controlling finite transition systems (such as T_c) from specifications given as LTL formulas. In this section, we show that a control strategy generated for T_c can be adapted to the infinite $T_{\mathscr{W}}$, while the satisfaction of LTL formulas by the closed loop systems is preserved, which completes the solution to Problem 4.

Definition 25 (PWA control strategy): A control strategy (X_0^c, Ω^c) for T_c can be translated into a control strategy (X_0, Ω) for $T_{\mathscr{W}}$ as follows. The initial set $X_0^c \subseteq X_c$ gives the initial set $X_0 = \bigcup_{l \in X_0^c} \mathbf{X}_l \subseteq X_{\mathscr{W}}$. Given a finite sequence of states $x_0 \ldots x_k$ where $x_0 \in X_0$, the control function is defined as $\Omega(x_0 \ldots x_k) = \Omega^c(o_{\mathscr{W}}(x_0) \ldots o_{\mathscr{W}}(x_k))$.

Proposition 4: Given a control strategy (X_0^c, Ω^c) for T_c translated as a control strategy (X_0, Ω) for $T_{\mathscr{W}}$, $\mathscr{L}_{T_{\mathscr{W}}}(X_0, \Omega) \subseteq \mathscr{L}_{T_c}(X_0^c, \Omega^c)$, which implies that if $T_c(Q_0^c, \Omega^c)$ satisfies an arbitrary LTL formula ϕ, then so does $T_{\mathscr{W}}(X_0, \Omega)$.

The overall solution to Problem 4 consists of constructing the control transition system T_c (Section VI-B), finding a satisfying control strategy for T_c and adapting it to the original $T_{\mathscr{W}}$, or equivalently PWA system (Definition 25), which from Proposition 4 guarantees the correctness of the solution. It is important to note that a control strategy generated using this approach is robust with respect to knowledge of the exact state of the system (*i.e.*, the control strategy depends on the observation of a state rather than the state itself). In addition, the control strategy is robust with respect to perturbations in the applied input bounded by ε, which can be used as a tuning parameter.

D. Conservatism and Stuttering Behavior

In Section V we described a solution to the problem of controlling a finite and possibly nondeterministic transition system from LTL specifications. In order to generate a control strategy for an infinite transition system such as $T_{\mathscr{W}}$ (Problem 4) we described the construction of a finite control abstraction T_c in Section VI-B. However, due to *spurious trajectories* (*i.e.*, trajectories of T_c not present in $T_{\mathscr{W}}$) we cannot guarantee that a control strategy will be found for T_c even if one exists for $T_{\mathscr{W}}$ and therefore, the overall method is conservative. In the following, we present an approach for reducing this conservatism. We characterize *stutter* steps as a specific class of spurious trajectories, which we introduce through Example 15 and Figure 15.

Example 15: Assume that a constant input $uuu\ldots$ produces a trajectory $x_1x_2x_3x_4\ldots$ in $T_{\mathscr{W}}$ where $o(x_1) = l_1, o(x_2) = o(x_3) = l_2, o(x_4) = l_3$ (Figure 15-A). The corresponding word $l_1l_2l_2l_3\ldots$ is a trajectory of T_c (*i.e.*, $l_1, l_2, l_3 \in X_c$) and from the construction described in Section VI-B it follows that $l_2 \in \delta_c(l_1, u)$ and $\{l_2, l_3\} \subseteq \delta_c(l_2, u)$ (Figure 15-B). Then, there exists a trajectory of T_c that remains infinitely in state $l_2 \in X_c$ under input u, which is not necessarily true for $T_{\mathscr{W}}$. Such spurious trajectories do not affect the correctness of a control strategy but increase the overall conservativeness of the method. We address this by characterizing *stuttering inputs*, which guarantee that the system will leave a state eventually, rather than in a single step, and using this additional information during the construction of the control strategy for T_c.

Definition 26 (Stuttering inputs): Given a state $l \in X_c$ and a set of states $C \in 2^{X_c}$, the set of inputs U_l^C is *stuttering* if and only if $l \in C$ and for all input words $u_0u_1\ldots$, where

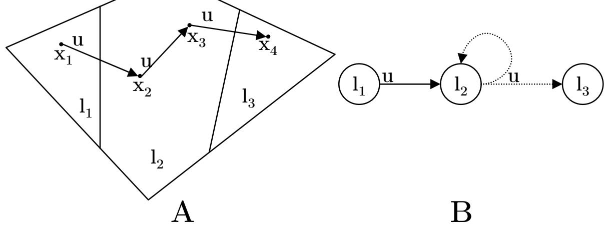


Fig. 15. A trajectory remaining forever in state l_2 exists in the finite abstraction **(B)**, although such a behavior is not necessarily possible in the concrete system **(A)**

$u_i \in U_l^C$, there exists a finite $k > 1$ such that the trajectory $x_0x_1\ldots$ produced in $T_{\mathscr{W}}$ by the input word satisfies $o(x_i) = l$ for $i = 1, \ldots, k-1$ and $o(x_k) = l' \in C, l' \neq l$.

Using Definition 26 we identify a stuttering subset $\Sigma_c^{ls} \subseteq \Sigma_c^l$ of the inputs available at a state $l \in X_c$. Let $u = c(U_l^C) \in \Sigma_c^l$ for some $C \in 2^{X_c}$ be an input of T_c computed as described in Section VI-B. Then $u \in \Sigma_c^{ls}$ if and only if U_l^C is stuttering. Note that a transition $\delta_c(l, u) = C$ from a state $l \in X_c$ where u is stuttering is always nondeterministic (*i.e.*, $|C| > 1$) and contains a self loop (*i.e.*, $l \in C$) but the self loop cannot be taken infinitely in a row (*i.e.*, a trajectory of $T_{\mathscr{W}}$ cannot remain infinitely in region $\mathbf{X}_l$ under input word $uuu\ldots$). An input $u \in \Sigma_c^{lu} = \Sigma_c^l \setminus \Sigma_c^{ls}$ induces a transition $\delta_c(l, u) = C$ where: (1) when $C = \{l\}$ trajectories of T_c and $T_{\mathscr{W}}$ produced by input word $uuu\ldots$ remain infinitely in state l and region $\mathbf{X}_l$, respectively, (2) when $l \notin C$ trajectories of T_c and $T_{\mathscr{W}}$ leave state l and region $\mathbf{X}_l$, respectively in one step under input u, (3) when $\{l\} \subset C$ trajectories of $T_{\mathscr{W}}$ produced by input word $uuu\ldots$ can potentially remain in region $\mathbf{X}_l$ infinitely. Although in case (3) it is also possible that trajectories of $T_{\mathscr{W}}$ produced by input word $uuu\ldots$ leave region $\mathbf{X}_l$ in finite time, we have to be conservative in order to guarantee the correctness of the control strategy.

Note that our definition of stuttering in Definition 26 requires that T_c leaves a state after a finite number of transitions are taken under the same stuttering input and therefore an infinite stutter cycle is never possible. Second, we identify a set of stuttering inputs rather than constructing T_c as a time abstract system. While we only characterize spurious infinite self loops (*i.e.*, cycles of length 1), in general, it is possible that cycles of arbitrary length are spurious in T_c. Considering higher order cycles is computationally challenging and decreases the conservativeness of the approach only for very specific cases, while spurious self loops are commonly produced during the construction of T_c and can be identified or constructed through polyhedral operations as described in the following (Propositions 5 and 6).

Proposition 5: Given a state $l \in X_c$ and a set of states $C \in 2^{X_c}$, input region U_l^C is stuttering if and only if $l \in C$ and $\vec{0} \notin \mathrm{hull}\{(A_l - I)v_x + B_l v_u + c_l \mid \forall v_x \in V(\mathbf{X}_l), \forall v_u \in V(U_l^C)\}$, where hull denotes the convex hull, $V(.)$ is the set of vertices and I is the identity matrix.

The strategy from Proposition 5 provides a computational characterization of stuttering input regions. In general, however, it is possible that an input region U_l^C cannot be identified as stuttering but a stuttering subset $\hat{U}_l^C \subset U_l^C$ can be identified. Then, if such a subset is "large enough" (*i.e.,* $r(\hat{U}_l^C) > \varepsilon$) it can be used in T_c and allow more general control strategies. In Proposition 6 we describe the computation of such stuttering subsets.

Proposition 6: Given an arbitrary $a \in \mathbb{R}^N$, the input region $\hat{U}_l^C = \{u \in U_l^C \mid \forall v \in V(\mathbf{X}_l), a^T B_l u > -a^T(A_l - I_N)v - c_l\}$, where $l \in C$ is always stuttering.

Although Proposition 6 is valid for an arbitrary $a \in \mathbb{R}^N$, the volume of the stuttering subset $\hat{U}_l^C \subset U_l^C$ depends on a. Since only "large enough" input regions are considered in T_c (see Algorithm 6), a should be chosen in such a way that the radius $r(\hat{U}_l^C)$ is maximized.

The control algorithms we discussed in Section V-A can be adapted to handle the additional information about stuttering inputs captured in T_c, while the correctness and completeness of the control strategy computation for the product automaton P is still guaranteed. P is constructed as in Section V-A and therefore it naturally inherits the partitioned input set $\Sigma_c^l = \Sigma_c^{ls} \cup \Sigma_c^{lu}$ for each state $l \in X_c$. Going back to the Rabin game interpretation of the control problem discussed in Section V-A, we need to account for the fact that the adversary cannot take transitions under the same stuttering input infinitely many times in a row. As a result, the construction of the control strategy is still performed using the same algorithm and only the computations of the direct attractors from Definitions 17 and 18 are modified as follows (the notation used in the rest of this section was introduced in Section V).

Let $l \in X_c$ and $u \in \Sigma_c^{ls}$ be a state and a stuttering input of T_c (Definition 26). We are interested in *edge* (s,u,s') of transition $\delta_P(s,u) = S'$, where $\alpha(s) = l$ and $s' \in S'$ (here $\alpha()$ is the projection of states from P to states of T_c. Edge (s,u,s') is called *u-nontransient* edge if $\alpha(s) = \alpha(s') = l$ and *transient* otherwise. Note that, even though (l,u,l) is a self loop in T_c, (s,u,s') is not necessarily a self loop in P. In addition, since there is at most one self loop at a state $l \in X_c$ and R is deterministic, there is at most one u-nontransient edge leaving state s.

We refer to a sequence of edges $(s_1,u_1,s_2)(s_2,u_2,s_3)\ldots(s_{n-1},u_{n-1},s_n)$, where $s_i \neq s_j$ for any $i,j \in \{1,\ldots,n\}$ as a *simple path*, and to a simple path $(s_1,u_1,s_2)\ldots(s_{n-1},u_{n-1},s_n)$ followed by (s_n,u_n,s_1) as a *cycle*. We can observe that any sequence of u-nontransient edges (*i.e.,* a run of the product automaton, or its finite fragment) is of one of the following shapes: a cycle (called a *u-nontransient cycle*), a lasso shape (a simple path leading to a u-nontransient cycle), or a simple path ending at a state where the input u is not available at all. Informally, the existence of a stuttering self loop in a state l under input u in T_c means that this self loop cannot be followed infinitely many times in a row. Similarly, any u-nontransient cycle in the product graph cannot be followed infinitely many times in a row without leaving it. This leads us to the new definitions of protagonist's and adversary's direct attractor.

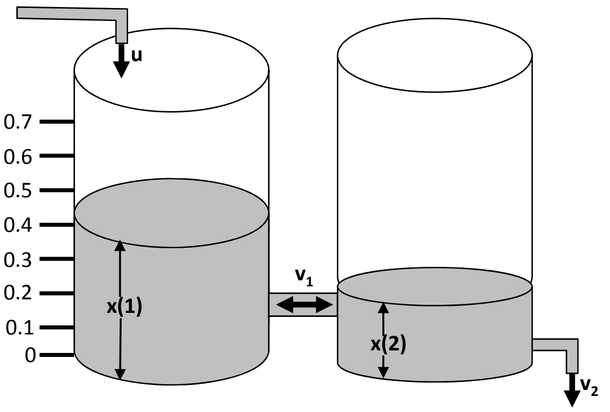


Fig. 16. A two-tank system. Water is drained at a constant rate from tank 2 though valve v_2, while tank 1 is filled at a rate that is controlled externally. Water can also flow in either direction, from the tank with more water to the one with less, through valve v_1, which is opened only if submerged.

Definition 27 (Modified protagonist's direct attractor): The protagonist's direct attractor of S', denoted by $\mathsf{A}_P^1(S')$, is the set of all states $s \in S_P$, such that there exists an input u satisfying

(1) $\delta_P(s,u) \subseteq S'$, or
(2) s lies on a u-nontransient cycle, such that each state s' of the cycle satisfies that $s'' \in S'$ for all transient edges (s',u,s'').

In other words, the protagonist can enforce a visit to S' also by following a u-nontransient cycle finitely many times and eventually leaving it to S'.

Definition 28 (Modified adversary's direct attractor): The adversary's direct attractor of S', denoted by $\mathsf{A}_S^1(S')$, is the set of all states $s \in S_P$, such that for each input u there exists a state s' such that

(1) $s' \in \delta_P(s,u) \cap S'$, and
(2) s' does not lie on a u-nontransient cycle.

In other words, the adversary cannot enforce a visit to S' via an edge of a u-nontransient cycle. This edge can be taken only finitely many times in a row and eventually a different edge under input u has to be chosen.

By identifying stuttering inputs during the construction of the control transition system T_c (Proposition 5 and Proposition 6) and modifying the approach from Section V-A to handle this additional information during the construction of a control strategy for T_c (Definitions 27 and 28), we can reduce the conservatism associated with the overall method. Even so, our solution to Problem 4 remains conservative but it is important to note that the only source of conservativeness is the construction of T_c — the solution to the LTL control problem for T_c is complete.

Example 16: We consider the problem of controlling the two-tank system shown in Figure 16. The system has two state variables ($N = 2$) that represent the water levels in the two tanks and range in $(0, 0.7)$. It has one control dimension ($M = 1$) representing the inflow rate, which ranges in $(0, 5e^{-4})$. The state space of the system is partitioned into 49 rectangular regions (*i.e.,* $L = 1,\ldots,49$) by 7 evenly spaced thresholds along each dimension. These thresholds signify that we can only detect whether the water level in each tank is above or below the marks at $0.1, 0.2, \ldots, 0.7$ (see Figure 17(a)). Valve v_1 is opened only if submerged (*i.e.,* if the water level in either tank is above 0.2 — the height of the valve) and, therefore, the valve is closed when the system is in regions $1,2,8,$ and 9 and opened otherwise (see Figures 16 and 17(a)). Discrete-time, linear equations describing the dynamics of the system in each mode are derived using a 5 sec. time step and $1.54e^{-2}\ m^2$, $1e^{-4}\ m^2$, and $2.125e^{-5}\ m^2$ as the cross sectional areas of the two tanks, valve v_1, and valve v_2, respectively. The dynamics of the system for each region $l \in L$ are given by

$$A_l = \begin{cases} \begin{bmatrix} 1 & 0 \\ 0 & 0.9635 \end{bmatrix} & \text{for } l = 1,2,8,9 \\ \begin{bmatrix} 0.8281 & 0.1719 \\ 0.1719 & 0.7916 \end{bmatrix} & \text{otherwise,} \end{cases}$$

$$B_l = [324.6753, 0]^T, c_l = [0,0]^T \text{ for } l = 1,\ldots,49.$$

We seek a control strategy for the system guaranteeing the satisfaction of a specification, expressed informally as "whenever tank 2 is empty, it will eventually get filled up". To formalize this specification we define the sub-formulas $\phi_1 =$ "the level of tank 2 is below 0.1" (*i.e.,* "tank 2 is empty") and $\phi_2 =$ "the level of tank 2 is above 0.4" (*i.e.,* "tank 2 is full"), which can be expressed as disjunctions of regions from L as $\phi_1 = 1 \vee \ldots \vee 7$ and $\phi_2 = 29 \vee \ldots \vee 49$. The above specification translates to the following LTL formula:

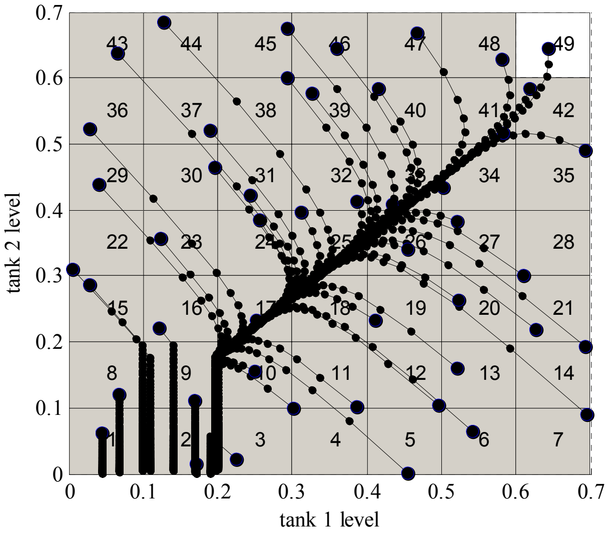
$$\phi = \Box(\phi_1 \Rightarrow \Diamond\phi_2).$$

Satisfying control strategies were found from all regions expect 49 when the required robustness was set to $\varepsilon = 5e^{-6}$ (Figure 17(a)). If stuttering inputs are not characterized as described in Section VI-D, satisfying control strategies are identified for regions $29, \ldots, 48$ only. A simulated trajectory of the closed loop system is shown in Figure 17(b).

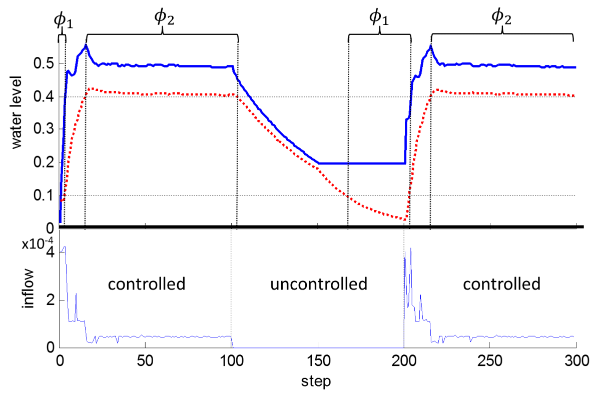
E. Complexity

As our proposed solution to Problem 4 consists of (1) the construction of the control transition system T_c and (2) the generation of a control strategy for T_c, the overall computational complexity is the cumulative complexity of the two parts. The computation of T_c involves enumerating all subsets of L at any element of L, which gives $O(|L| \cdot 2^{|L|})$ iterations in the worst case, although in practice this can be reduced (see Equation (37)). At each iteration, polyhedral operations are performed, which scale exponentially with N, the size of the continuous state space. The characterization of stuttering inputs described in this chapter checks each element from Σ_c through polyhedral operations.

The overall complexity of the control strategy synthesis for T_c is $\mathcal{O}(k!n^k)$, where n is the size of the product automaton and k is the number of pairs in the Rabin condition of the product automaton (see Section V). The modifications in the computation of the direct attractor we made in order to adapt the algorithm to deal with stuttering behavior do not change the overall complexity. Note that, in general, Rabin games are NP-complete, so the exponential complexity with respect to k is not surprising. However, LTL formulas are usually translated into Rabin automata with very few tuples in their acceptance condition.



(a) Simulated trajectories of the uncontrolled system.



(b) A simulated trajectory of the closed-loop system.

Fig. 17. Simulated trajectories of the uncontrolled and closed-loop water tank system from Figure 16. Initial conditions are shown as blue circles and regions are labeled only by their indexes in (a). Control strategies guaranteeing the satisfaction of specification $\phi = \Box(\phi_1 \Rightarrow \Diamond\phi_2)$ are found from all shaded regions in (a). The water levels of tanks 1 and 2 are shown respectively as a blue (solid) and a red (dashed) line in (b) and the trajectory is guaranteed to satisfy specification ϕ. See Example 16 for additional details.

VII. Conclusion

In this tutorial paper, we focused on formal synthesis problems for discrete-time piecewise affine systems from specifications given as Linear Temporal Logic (LTL) formulas over linear predicates in the state variables. We provided a detailed, self-contained solution to this problem. Specifically, we introduced transition systems as general models for finite and infinite systems, together with simulations and bisimulations relations used to construct abstractions. We then defined and solved the LTL synthesis problem for a finite non-deterministic system. Finally, we provided a two-step, conservative solution to the LTL synthesis problem for a discrete-time piecewise affine system. The first consists of the construction of a finite abstraction. The second step is solving a game in the abstraction.

References

[1] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*. Springer, 2016, to appear.

[2] A. Arnold, *Finite transition systems: semantics of communicating systems*. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1994.

[3] E. M. M. Clarke, D. Peled, and O. Grumberg, *Model checking*. MIT Press, 1999.

[4] C. A. R. Hoare, *Communicating sequential processes*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1985.

[5] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, pp. 231–274, June 1987. [Online]. Available: http://portal.acm.org/citation.cfm?id=34884.34886

[6] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[7] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2008.

[8] T. Kropf, *Introduction to Formal Hardware Verification: Methods and Tools for Designing Correct Circuits and Systems*, 1st ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.

[9] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, pp. 971–984, 2000.

[10] G. J. Pappas, "Bisimilar linear systems," *Automatica*, vol. 39, no. 12, pp. 2035–2047, 2003.

[11] P. Tabuada and G. Pappas, "Model checking LTL over controllable linear systems is decidable," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, O. Maler and A. Pnueli, Eds. Springer-Verlag, 2003, vol. 2623, pp. 498–513.

[12] B. Yordanov and C. Belta, "Formal analysis of discrete-time piecewise affine systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 12, pp. 2834 –2840, 2010.

[13] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta, "Temporal logic control of discrete-time piecewise affine systems," *IEEE Transactions on Automatic Control*, vol. 57, pp. 1491–1504, 2012.

[14] ——, "Formal analysis of piecewise affine systems through formula-guided refinement," *Automatica*, vol. 49, no. 1, pp. 261–266, 2013.

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

[16] ——, "Reachability analysis of multi-affine systems," *Transactions of the Institute of Measurement and Control, special issue on Hybrid Systems*, 2009.

[17] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.

[18] R. Alur, *Principles of Cyber-Physical Systems*. MIT Press, 2015.

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

[20] E. Amir and A. Chang, "Learning partially observable deterministic action models," *Journal of Articial Intelligence Research*, vol. 33, pp. 349–402, 2008.

[21] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[22] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[23] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee, "POMDPs for robotic tasks with mixed observability," in *Robotics: Science and Systems*, 2009.

[24] R. Milner, *Communication and concurrency*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[25] K. G. Larsen and A. Skou, "Bisimulation through probabilistic testing," *Information and Computation*, vol. 94, pp. 1–28, 1991.

[26] A. Girard and G. Pappas, "Approximate bisimulation relations for constrained linear systems," *Automatica*, vol. 43, pp. 1307 – 1317, 2007.

[27] G. Pola, A. Girard, and P. Tabuada, "Approximately bisimilar symbolic models for nonlinear control systems," *Automatica*, vol. 44, no. 10, pp. 2508–2516, 2008.

[28] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science: Formal Models and Semantics*, J. van Leeuwen, Ed. North-Holland Pub. Co./MIT Press, 1990, vol. B, pp. 995–1072.

[29] L. D. Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Foundations of Software Technology and Theoretical Computer Science*, ser. LNCS. Springer-Verlag, 1995, pp. 499–513.

[30] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time markov chains," *IEEE Trans. Softw. Eng.*, vol. 29, no. 6, pp. 524–541, 2003.

[31] P. Zuliani, A. Platzer, and E. M. Clarke, "Bayesian Statistical Model Checking with Application to Simulink/Stateflow Verification," in *International Conference on Hybrid systems: Computation and Control (HSCC) 2010*, 2010, pp. 243–252. [Online]. Available: http://doi.acm.org/10.1145/1755952.1755987

[32] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pp. 71–76, 2004.

[33] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real- Time Systems*, vol. 2, no. 4, p. 255?299, 1990.

[34] M. O. Rabin, "Decidability of second-order theories and automata on infinite trees," *Trans. Amer. Math. Soc.*, vol. 141, pp. 1–35, 1969.

[35] E. A. Emerson, "Automata, tableaux and temporal logics (extended abstract)," in *Proceedings of the Conference on Logic of Programs*, 1985, pp. 79–88.

[36] N. Piterman and A. Pnueli, "Faster solutions of rabin and streett games," in *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 275–284.

[37] F. Horn, "Streett Games on Finite Graphs," 2005, in the *2nd Workshop on Games in Design and Verification (GDV)*.

[38] J. Tůmová, B. Yordanov, C. Belta, I. Černá, , and J. Barnat, "A symbolic approach to controlling piecewise affine systems," in *Proceedings of the 49th IEEE Conference on Decision and Control*, Atlanta, GA, Dec. 2010, pp. 4230–4235. [Online]. Available: pdf/cdc10_1.pdf

[39] J. R. Büchi, "On a decision method in restricted second order arithmetic," in *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science 1960*, E. N. et al., Ed. Stanford, CA: Stanford University Press, 1962, pp. 1–12.

[40] W. Thomas, "Infinite games and verification," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, E. Brinksma and K. Larsen, Eds. Springer Berlin / Heidelberg, 2002, vol. 2404, pp. 58–65.

[41] M. Kloetzer and C. Belta, "Dealing with non-determinism in symbolic control," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 4981. Springer Berlin / Heidelberg, 2008, pp. 287–300.

[42] M. Antoniotti and B. Mishra, "The supervisor synthesis problem for unrestricted ctl is np-complete," 1995.

[43] E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Science of Computer Programming*, vol. 2, no. 3, p. 241266, 1982.

[44] S. Jiang and R. Kumar, "Supervisory control of discrete event systems with CTL* temporal logic specifications," *SIAM Journal on Control and Optimization*, vol. 44, no. 6, pp. 2079–2103, 2006.

[45] A. Ulusoy and C. Belta, "Receding horizon temporal logic control in dynamic environments," *The International Journal of Robotics Research*, vol. 33, no. 12, pp. 1593–1607, 2014.

[46] M. Svorenova, I. Cerna, and C. Belta, "Optimal temporal logic control for deterministic transition systems with probabilistic penalties," *IEEE Transactions on Automatic Control*, vol. 60, no. 6, pp. 1528–1541, 2015.

[47] X. C. Ding, M. Lazar, and C. Belta, "Ltl receding horizon control for finite deterministic systems," *Automatica*, vol. 50, no. 2, pp. 399–408, 2014.

[48] K. Chatterjee, A. Gaiser, and J. Kretinsky, "Automata with generalized rabin pairs for probabilistic model checking and ltl synthesis," in *Proc. of CAV*, Saint Petersburg, Russia, 2013.

[49] X. Ding, S. L. Smith, C. Belta, and D. Rus, "Optimal control of markov decision processes with linear temporal logic constraints," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1244–1257, 2014.

[50] M. Svorenova, I. Cerna, and C. Belta, “Optimal control of mdps with temporal logic constraints,” in *52nd IEEE Conference on Decision and Control (CDC)*, Firenze, Italy, 2013.

[51] W. P. M. H. Heemels, B. D. Schutter, and A. Bemporad, “Equivalence of hybrid dynamical models,” *Automatica*, vol. 37, no. 7, pp. 1085–1091, 2001.

[52] A. Garulli, S. Paoletti, and A. Vicino, “A survey on switched and piecewise affine system identification,” *IFAC Proceedings Volumes*, vol. 45, no. 16, pp. 344 – 355, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667015379751

[53] A. L. Juloski, W. P. M. H. Heemels, G. Ferrari-Trecate, R. Vidal, S. Paoletti, and J. H. G. Niessen, “Comparison of four procedures for the identification of hybrid systems,” in *Proceedings of the 8th International Conference on Hybrid Systems: Computation and Control (HSCC)*, ser. LNCS, 2005, pp. 354–369.

[54] B. Yordanov and C. Belta, “Temporal logic control of discrete-time piecewise affine systems,” in *Proceedings of the 48th IEEE Conference on Decision and Control*, Shanghai, China, Dec. 2009, pp. 3182 –3187. [Online]. Available: pdf/cdc09.pdf

[55] P. Tabuada and G. Pappas, “Linear time logic control of discrete-time linear systems,” vol. 51, no. 12, pp. 1862–1877, 2006.

[56] S. Coogan, E. A. Gol, M. Arcak, and C. Belta, “Traffic network control from temporal logic specifications,” *IEEE Transactions on Control of Network Systems*, vol. 3, no. 2, p. 162 ? 171, 2016.

[57] A. Tiwari and G. Khanna, “Series of Abstractions for Hybrid Automata,” in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, C. Tomlin and M. Greenstreet, Eds. Springer Berlin / Heidelberg, 2002, vol. 2289, pp. 425–438.

[58] O. Pãun and M. Chechik, “On Closure Under Stuttering,” *Formal Aspects of Computing*, vol. 14, no. 4, pp. 342–368, Apr. 2003.

[59] D. A. Peled and T. Wilke, “Stutter-invariant temporal properties are expressible without the next-time operator,” *Information Processing Letters*, vol. 63, no. 5, pp. 243–246, Sept. 1997.

[60] N. Ozay, J. Liu, P. Prabhakar, and R. M. Murray, “Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems,” in *American Control Conference*, June 2013, pp. 6237–6244.

[61] E. A. Gol, X. C. Ding, M. Lazar, and C. Belta, “Finite bisimulations for switched linear systems,” in *IEEE Conference on Decision and Control (CDC) 2012*, Maui, Hawaii, 2012.

[62] E. A. Gol, M. Lazar, and C. Belta, “Language-guided controller synthesis for discrete-time linear systems,” in *15th International Conference on Hybrid Systems: Computation and Control (HSCC)*, Beijing, China, 2012.