Dealing with Nondeterminism in Symbolic Control*

Marius Kloetzer and Calin Belta

Center for Information and Systems Engineering
Boston University
15 Saint Mary's Street, Boston, MA 02446
{kmarius,cbelta}@bu.edu

Abstract. Abstractions (also called symbolic models) are simple descriptions of continuous and hybrid systems that can be used in analysis and control. They are usually constructed in the form of transition systems with finitely many states. Such abstractions offer a very attractive approach to deal with complexity, while at the same time allowing for rich specification languages. Recent results show that, through the abstraction process, the resulting transition systems can be nondeterministic (*i.e.,* if an input is applied in a state, several next states are possible). However, the problem of controlling a nondeterministic transition system from a rich specification such as a temporal logic formula is not well understood. In this paper, we develop a control strategy for a nondeterministic transition system from a specification given as a Linear Temporal Logic formula with a deterministic Büchi generator. Our solution is inspired by LTL games on graphs, is complete, and scales polynomially with the size of the Büchi automaton. An example of controlling a linear system from a specification given as a temporal logic formula over the regions of its triangulated state space is included for illustration.

1 Introduction

In control problems, trajectories of "complex" mathematical models, such as systems of differential equations, are usually checked against "simple" specifications, such as stability of equilibria and set invariance. In formal verification, "rich" specifications, such as formulas of temporal logics, are checked against "simple" models of software programs and digital circuits, such as (finite) transition graphs. There has been a lot of interest lately in developing theoretical frameworks and computational tools for bridging in this gap, and therefore allowing for specifying the properties of continuous and hybrid systems in a rich language, with automatic verification and controller synthesis. Most of the existing approaches are centered at the concept of abstraction, *i.e.,* the process through which a system with infinitely many states (such as a control system in continuous space and time) is mapped to a system with finitely many states, called symbolic, or abstract model. Roughly, the abstract model can be seen as a transition graph, whose states label "equivalent" sets of states of the initial system.

* This work is partially supported by NSF CAREER 0447721 and NSF 0410514 at Boston University.

M. Egerstedt and B. Mishra (Eds.): HSCC 2008, LNCS 4981, pp. 287–300, 2008.

The abstract model can be either equivalent with the initial system with respect to the satisfaction of the specification, or it can provide an approximation, with the guarantee that the satisfaction of the specification for the abstract model is sufficient for the satisfaction of the specification by the initial system. Equivalent abstractions are based on the notion of bisimulation [1], while sufficient abstractions can be derived using simulation relations. The class of systems for which equivalent finite models exist include systems with very simple continuous dynamics, such as timed automata [2], multirate automata [3], rectangular automata [4], or systems with more complex continuous dynamics but simpler discrete dynamics, such as o-minimal hybrid systems [5]. More recent results provide conditions for the existence of equivalent abstractions for discrete-time continuous-space linear systems [6] and for more general systems through a relaxed notion of approximate bisimulation [7,8]. Recent works on constructing sufficient abstractions focus on systems with linear dynamics and polyhedral partitions [9] and systems with polynomial dynamics and partitions given by semi-algebraic sets [10]. In these works, the construction of sufficient or equivalent abstractions (if they exist) is expensive, and involves either the integration of vector fields [9] or quantifier elimination for real closed fields and theorem proving [10].

There are two classes of systems for which checking the existence of equivalent abstractions and the construction of sufficient abstractions can be reduced to polyhedral operations only [11]: affine systems with simplicial partitions (*e.g.,* triangulations in the 2D case) and multi-affine systems with rectangular partitions. Roughly, such constructions are possible because necessary and sufficient conditions for the existence of controllers driving all initial states of an affine (multi-affine) system in a simplex (rectangle) through a facet in finite time and for making a simplex (rectangle) an invariant can be reduced to checking the non-emptiness of polyhedral sets [12,13]. If in a simplicial (rectangular) partition of the state space of an affine (multi-affine) system, feedback controllers can be designed such that all states either stay inside or leave through a facet (to a neighbor region), then the corresponding quotient transition system is an equivalent abstraction (bisimulation quotient). Moreover, this finite transition system is deterministic, since an applied control uniquely determines the next state. If the control specification for the initial system is given as a Linear Temporal Logic (LTL) formula over the regions of the partitioned state space, then the problem reduces to controlling a deterministic transition system from an LTL formula over its states. This problem is relatively easy, since it can be solved by adapting standard tools from LTL model checking [14]. We proposed a solution in [15], and used it to develop a fully automated procedure for control of linear systems from specifications given as arbitrary LTL formulas over arbitrary linear predicates in the state variables.

This paper is motivated by recent results [16,17] extending the work from [12,13]. Specifically, in [16], the authors showed that, for an affine system in a simplex, even though a controller driving all states through a facet (*i.e.,* to a neighbor) might fail to exist, controllers driving the system through a set of facets (*i.e.,* to a set of neighbors) might be found. Similar results were proved for multi-affine systems and rectangles in [17]. While reducing the conservativeness introduced through the abstraction process, these results raise a new problem: since a controller does not guarantee a transition to exactly one neighbor, the abstract transition system is non-deterministic. On the other

hand, the problem of controlling a non-deterministic transition system from a rich specification such as an LTL formula over its states is currently not solved.

In this paper, we focus on specifications given as formulas of a fragment of LTL [14] for which the corresponding languages are generated by deterministic Büchi automata. We propose a solution inspired from (infinite) LTL games [18,19,20], which are played by two players on a graph. Roughly said, we generalize this problem to transition systems with inputs, and treat non-determinism as an adversary. The solution is presented in the form of a feedback automaton, which at each step reads the current state of the transition system and generates the applied control. We approached this problem in our previous work [21], where we mapped it to a classical LTL game played on a modified transition system by assigning its states to the two players. As opposed to [21], the solution that we propose here is *complete*, in the sense that we find a solution if one exists. The algorithms proposed in this paper were implemented as a user-friendly software tool under Matlab, which is freely downloadable from `http://iasi.bu.edu/~software/nondet.htm`.

2 Case Study

To motivate the problem and illustrate our approach, we consider an example of controlling an affine system from a specification given as a temporal logic statement about the reachability of simplices in a triangulation of its state space. Consider the following affine system:

$$\dot{x} = \begin{bmatrix} -0.4 & 0.2 \\ 0.5 & -0.8 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u + \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix}, \tag{1}$$

where the state and controls are restricted to rectangular sets $x \in [2, 10] \times [1, 7]$ and $u \in [-1, 1] \times [-1, 1]$, respectively. Assume the (planar) state space of the system is triangulated as shown in Fig. 1 (a).

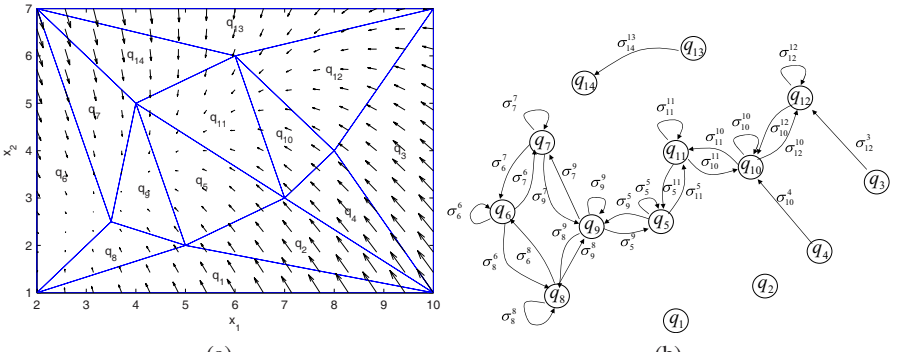

Fig. 1. (a) Triangular partition of the planar state space of system (1) and the vector field corresponding to the uncontrolled system. (b) The deterministic transition system modelling controllers driving all states in a simplex through a facet or making a simplex an invariant (σ_j^i with $i \neq j$ is a feedback controller for q_i guaranteeing exit to q_j in finite time; σ_i^i is a feedback controller making q_i an invariant.

Assume we want to find initial states and control strategies for system (1) such that all the trajectories of the closed loop system satisfy the following specification:

$$\text{“}eventually\ visit\ q_6\ and\ q_{12},\ in\ any\ order\text{”} \tag{2}$$

In other words, it is desired that the trajectories of the closed loop system evolve in the triangulated environment such that, at some point in the future, the regions labelled by q_6 and q_{12} (not necessarily in this order), are reached. Note that this specification contains temporal (“eventually”) and logical (“and”) information. Temporal logics [14] offer formal frameworks for such temporal and logical statements. Specifically, in our case, the specification translates to the following Linear Temporal Logic (LTL) formula over the set of symbols $\{q_1, \ldots, q_{14}\}$:

$$\phi = \Diamond q_6 \wedge \Diamond q_{12}, \tag{3}$$

where “$\Diamond$” means “eventually” and “$\wedge$” is the well known notation for “and”. Therefore, the problem translates to controlling the affine system (1) from a specification given as the temporal logic formula (3) over triangles in its partitioned state space.

This problem can be seen as a particular case of the framework we developed in [15], where an arbitrary linear system was controlled from LTL formulas over arbitrary linear predicates in its state variables. In short, the computational tool developed in [15] consists of the following steps: (i) construct a polyhedral partition of the state space using the linear predicates in the specification, (ii) for each of the polytopes in the partition, find feedback controllers making the polytope an invariant, and, for each of its facets, find feedback controllers driving all the initial states in the polytope through the facet (to a neighbor polytope) in finite time, (iii) arrange the results of the previous two steps in the form of a transition system (or transition graph), where the states (nodes) label the polytopes, and the transitions (included in the adjacency relations between polytopes) are labelled by the corresponding controllers designed in the previous step (a self transition corresponds to a controller making the corresponding polytope an invariant set), (iv) design a control strategy for the transition system constructed in the previous step, in the form of a hybrid system. For our example, the transition system obtained in step (iii) is shown in Figure 1 (b).

Since an applied control uniquely determines the next state, the transition system constructed above is deterministic (*e.g.*, the transition system in Figure 1 (b)). Therefore, step (iv) of the above procedure reduces to the problem of controlling a deterministic transition system from a specification given as an LTL formula over its states. One can find a solution to this problem by model checking the transition system with the negation of the formula using (off-the-shelf tools, such as SPIN and NuSMV). However, there is no control over the produced counterexample, which might be too long, or simply not implementable by the initial continuous system. To overcome this, in [15], we proposed another solution for the problem, which will be also briefly reviewed in Remark 3. The solution consists of the following steps: (a) construction of a Büchi automaton accepting the language satisfying the formula, (b) synchronization of the transition system with the Büchi automaton by taking their product, (c) finding a run in the product automaton that is implementable by the initial continuous system and is optimal with respect to a cost imposed by the particular application, and (d) finding

the control strategy in the form of the sequence of controls producing the run from the previous step. Step (d) is possible because, in a deterministic transition system, a run is uniquely determined by a sequence of applied controls.

By applying the procedure from [15] to our example, we find that the set of initial states for which control strategies can be designed such that all trajectories of the closed loop system satisfy the formula is given by the union of all triangles except for q_1, q_2, q_{13}, q_{14}. The solution is of course conservative, in the sense that, even though we label some triangles as not containing initial states for trajectories satisfying the formula, there might exist states in these triangles such that trajectories originating there satisfy the formula. There are two sources of conservativeness in this approach: (1) the initial states are treated as whole sets given by the initial triangles (no subpartition is performed), and (2) only controllers either driving all initial states to a facet or making a triangle an invariant are allowed. Related to the latter source of conservativeness, one can imagine that, even though controllers driving all initial states to a facet might fail to exist, controllers guaranteeing exiting through a set of facets might be found. For example, by using the techniques from [12], no feedback controllers can be found to drive the states in triangle q_1 through the separating facet with q_8, or through the separating facet with q_2. However, by using the more general conditions from [16], a feedback controller can be found which guarantees that all initial states in q_1 will eventually reach either q_8 or q_2. However, if such controllers are allowed, the quotient transition system becomes nondeterministic. For our example, the resulting transition system is shown in Figure 2 (b). By applying the general method for controlling nondeterministic transition systems proposed in this paper, coupled with feedback controllers as in [16], we show in Section 6 that control strategies producing trajectories satisfying the formula can be found for all initial states in $q_1, q_2, \ldots, q_{14}$. In other words, the conservatism from our previous method is considerably reduced.

3 Preliminaries

Throughout the paper, for a finite set A, we will use the notations $|A|$, A^ω, and 2^A to denote its cardinality, the set of all infinite words over A, and its power set (the set of all its subsets), respectively.

Definition 1 (Transition system). *A finite (nondeterministic) transition system is a tuple $T = (Q, \Sigma, \delta)$, where:*

- *Q is a finite set of states,*
- *Σ is a finite input alphabet,*
- *$\delta : Q \times \Sigma \to 2^Q$ is a (nondeterministic) transition function.*

For a given state $q \in Q$, the set of available (feasible) inputs is denoted by Σ_q (*i.e.,* Σ_q is the set of $\sigma_i \in \Sigma$ for which $|\delta(q, \sigma_i)| \geq 1$). An *input word* $\sigma \in \Sigma^\omega$ is denoted by $\sigma = \sigma_1\sigma_2\sigma_3 \ldots$ A *trajectory* or *run* of T produced by an input word σ starting from q is an infinite sequence $w \in Q^\omega$, $w = w_1w_2w_3 \ldots$ with the property that $w_1 = q$ and $\forall i \geq 1$, $w_{i+1} \in \delta(w_i, \sigma_i)$.

A formal definition of the syntax and semantics of Linear Temporal Logic (LTL) formulas is beyond the scope of this paper. Intuitively, an LTL formula over the set Q

is any "sensible" combination of elements from Q, logical operators $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication), $\Leftrightarrow$ (equivalence), and temporal operators $\bigcirc$ (next), $\mathcal{U}$ (until), $\Diamond$ (eventually), $\Box$ (always). The semantics of LTL formulas are given over runs of transition system T. For example, if the states Q of T are labels for regions in a partitioned state space of a control system, then the control specification "visit region q_1, then region q_2, and then go to final target q_3, while avoiding q_1" translates to formula

$$\Diamond(q_1 \wedge \Diamond(q_2 \wedge (\neg q_1)\mathcal{U}q_3)) \tag{4}$$

which is true for any run in which q_1 appears at some position, then q_2 appears, and then eventually q_3 appears, while q_1 does not appear before this happens.

For every LTL formula over Q, there exists a Büchi automaton (also called a generator of the LTL formula) accepting all and only the words satisfying it [22].

Definition 2 (Büchi automaton). *A Büchi automaton is a tuple $B = (S, S_0, Q, \delta_B, F)$, where:*

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

The semantics of a Büchi automaton is defined over infinite input words. Let $w = w_1 w_2 w_3 ..., w \in Q^\omega$, be an infinite input word of automaton B. We denote by $\mathcal{R}_\mathcal{B}(w)$ the set of all initialized runs of B that can be produced by w:

$$\mathcal{R}_\mathcal{B}(w) = \{r = s_1 s_2 s_3 ... |\, s_1 \in S_0,\ s_{i+1} \in \delta_B(s_i, w_i),\ \forall i \geq 1\} \tag{5}$$

Definition 3 (Büchi acceptance). *A word $w \in Q^\omega$ is accepted by the Büchi automaton B if and only if $\exists r \in \mathcal{R}_\mathcal{B}(w)$ so that $inf(r) \cap F \neq \emptyset$, where $inf(r)$ denotes the set of states appearing infinitely often in the run r.*

In words, an input word w is accepted by B if and only if there exists at least a run induced by w that visits F infinitely often.

Remark 1. Motivated by the particular control application, we use simplified definitions of transition system, Büchi automaton, and Büchi acceptance. We refer to [14] for more general definitions.

4 Problem Formulation and Approach

Problem 1. Given a transition system $T = (Q, \Sigma, \delta)$ and a Büchi automaton $B = (S, S_0, Q, \delta_B, F)$, find a set of initial states $Q_0 \subseteq Q$ and a control strategy for T such that all runs of T are accepted by B.

The Büchi automaton B from the above problem should be seen as a generator for an LTL formula ϕ over Q, which is the high-level control specification. Due to some complexity issues that go beyond the scope of this paper, in this work we assume that the Büchi automaton B from Problem 1 is *deterministic* (*i.e.,* $S_0 = \{s_0\}$ is a singleton, and $\delta_B : S \times Q \to S$ is a partial function). Intuitively, the current state of a deterministic Büchi automaton shows the progress towards the satisfaction of the LTL formula, and this information is used in developing the strategy from that point on. If a deterministic Büchi automaton does not exist for a given formula, the nondeterministic Büchi generator has to be translated into a more complicated type of deterministic automaton, like Muller or Rabin.

Remark 2. It is important to note at this point that, although for any LTL formula a generator Büchi automaton can be constructed, this automaton is in general nondeterministic, and cannot be always determinized [23]. However, the assumption that the Büchi generator is deterministic does not seem restrictive from an expressivity point of view [24], since most LTL formulas capturing control tasks (including the formulas in Eqn. (3), (4), as well as specifications like safety and liveness) belong to some isolated fragments of LTL, for which (partially-ordered) deterministic generators can be constructed [25].

Remark 3. If the transition system T were deterministic (*i.e.* $\delta : Q \times \Sigma \to Q$), then a solution to Problem 1 could be found by using an idea similar to model checking [14,15]: the product automaton $T \times B$ is computed and in this product automaton an accepting run with a specific structure is found and projected to a run of T. Since T is deterministic, a control strategy implementing the desired run can be constructed. This approach also works in the case of nondeterministic Büchi automata.

Problem 1 is related to the problem of controlling a discrete event system modelled as a transition system with inputs [26]. However, in this latter case, the specification is given as an ω-regular expression over inputs (rather than an LTL formula over states). In this paper, we propose a method inspired by the theory of LTL games. An LTL game is defined on a graph $G = (V, E)$ and it is played by two players: a protagonist and an adversary. The set of nodes (states) V is partitioned into a set of protagonist's states V_p, from which the protagonist can choose the next state, and a set of adversary's states V_a, from which the adversary chooses the next state [18]. A play consists of an infinite sequence of states resulted from an infinite sequence of transitions (edges) chosen by the two players. The specification for an LTL game is an LTL formula over the set of states V. A play is won by the protagonist if the produced run satisfies the LTL formula.

The protagonist has a winning strategy if, whenever the current state is in V_p, she manages to choose transitions such that she wins the current game, no matter what transitions the adversary chooses when the current state is in V_a. The goal of an LTL game is to find the set of initial states from where the protagonist has winning strategies and a winning strategy for plays starting in those initial states. The existing algorithms for solving LTL games are complete, in the following sense: when starting from the found set of initial states, the winning strategy guarantees that the protagonist wins the game, no matter how smart the adversary is, and when starting from any other state, the adversary has a strategy prohibiting the protagonist's winning [19].

Intuitively, we can think of control Problem 1 as an LTL game in which the adversary uses the non-determinism in the problem in the smartest way possible to prevent us from producing runs of T satisfying the formula. More precisely, while we have full control in choosing the current input of the transition system T in every state from Q, the adversary can choose the next state in the case when the chosen input produces nondeterministic transitions.

Maybe the simplest way of transforming our problem into a standard LTL game would be to partition the set of states Q in two sets: a protagonist's set Q_p, with each state having only deterministic outgoing transitions, and an adversary's set Q_a, with each state having at least one input producing non-deterministic transitions. Then, a graph with vertices Q can be easily constructed, where the adversary has full control in choosing any existing transition from states in Q_a. An algorithm for solving LTL games can be applied to this graph, and the resulting winning strategy (if any) can be adapted for the initial T as follows: in states from Q_p, the winning strategy (giving the next state to be reached) is easily mapped to the input producing the deterministic transition to the desired next state, and in states from Q_a any feasible input can be applied (since we gave full control to the adversary). Obviously, this strategy is conservative because we don't use our power of choosing inputs in every state, but instead we give all the transitions (inputs) from some states to the adversary. An attempt to reduce this conservatism can be as follows: first, place all states from Q in Q_p. Then, for each non-deterministic input in a node, replace the non-deterministic transitions with a deterministic one to a newly added state in Q_a, and assign the removed non-deterministic transitions to this new state. The solution is correct only for some LTL fragments, it is more complex, and it still cannot be proved to be complete.

In [21], Problem 1 was mapped to an LTL game for an augmented transition system obtained by splitting the states of T and by assigning them to the protagonist and the adversary. However, this procedure led to a conservative, incomplete solution. In the next section, we present a different approach, which is based on an adaptation of the LTL game algorithms, and which leads to a complete solution to Problem 1.

5 Solution to Problem 1

In this section, we show how the main steps involved in solving an LTL game [19] can be adapted to our problem. We first construct a product automaton P between the transition system T and the Büchi automaton B (Sect. 5.1). We then solve a Büchi game on P and find a set of initial states, together with a memoryless (positional) winning strategy (Sect. 5.2). Finally, the set of initial states and the winning strategy for T are obtained by projecting the initial states of P into Q and by adapting the winning strategy of P for T, respectively (Sect. 5.3). Unlike the winning strategy for P, the one corresponding to T will have memory.

5.1 Constructing the Product Automaton

Definition 4 (Product automaton). *The product automaton $P = T \times B$ between the nondeterministic transition system $T = (Q, \Sigma, \delta)$ and the deterministic Büchi automaton $B = (S, S_0, Q, \delta_B, F)$ is defined as the tuple $P = (S_P, S_{P0}, \Sigma, \delta_P, F_P)$, where:*

- $S_P = Q \times S$ *is the finite set of states,*
- $S_{P0} = Q \times S_0$ *is the set of initial states,*
- Σ *is the input alphabet,*
- $\delta_P : S_P \times \Sigma \to 2^{S_P}$ *is the transition function, defined as* $\delta_P((q,s),\sigma) = \{(q',s') \in S_P \mid q' \in \delta(q,\sigma) \text{ and } s' = \delta_B(s,q)\}$*, where* $(q,s) \in S_P$ *and* $\sigma \in \Sigma$*,*
- $F_P = Q \times F$ *is the set of accepting (final) states.*

The product automaton P is in fact a nondeterministic Büchi automaton with input alphabet Σ. Its acceptance condition is formulated as in Definition 3, but with respect to input words from Σ^ω. The product automaton in Definition 4 can be regarded as a match between the states and transitions of T and B. The transition function of P captures both the nondeterministic behavior of T and the way of deterministically tracking the progress towards the satisfaction on the LTL formula corresponding to B (infinitely visiting set F of B).

Let $w_P \in \Sigma^\omega$ be an accepted input of automaton P and let $r_P = (q_{i1}, s_{j1})\,(q_{i2}, s_{j2})\,(q_{i3}, s_{j3})...$ be a resulted run such that $inf(r_P) \cap F_P \neq \emptyset$. Then, a result from the model checking theory states that the projection of r_P to states in Q is a run $r_T = q_{i1}, q_{i2}, q_{i3}...$ of T accepted by B. Furthermore, a run of T accepted by B exists if and only if P has an accepted run. However, since T is nondeterministic, we cannot make sure that a certain run will be followed, as we could for the deterministic case mentioned in Remark 3. Therefore, our goal becomes to design a controller that applies to T inputs guaranteeing that any possible run will be accepted by B. Looking at P, this goal translates to designing a strategy of applying inputs to P such that any possible run r_P satisfies $inf(r_P) \cap F_P \neq \emptyset$.

This problem resembles a Büchi game, which is an intermediate step in solving a classical LTL game. The results from the next section can be seen as an extension of the solution to the Büchi game from partitioned graphs [18] to transition systems with inputs, where the protagonist can choose inputs and the adversary can choose the next state in nondeterministic transitions.

5.2 Solving a Büchi Game

As stated in the previous subsection, we want to apply inputs to P such that the subset of states F_P will be visited infinitely often. Whenever a nondeterministic transition is encountered, even though we are able to choose the input, the adversary will decide the next state, and we have to make sure that we will be able to accomplish the goal, no matter what state the adversary chooses. Eventually, by using a fixed-point strategy, we will be able to isolate a set of states $W_P \subseteq S_P$ (the winning region), from where we can guarantee infinitely many visits to F_P. An immediate adaptation of a result from the theory of LTL games [18] states that if a nonempty set W_P together with winning strategy of applying inputs exist, then there also exists a memoryless (positional) strategy, which applies a certain input in each state from W_P. In other words, a memoryless strategy will be a map $\pi_P : W_P \to \Sigma$.

In the following three definitions, which are adapted from [19], A is an arbitrary subset of the set of states S_P.

Definition 5 (Recurrent set). *The recurrent set of A, denoted by $\mathcal{R}(A)$, is defined as the set of all $s \in A$ from which there can be enforced infinitely many revisits to set A.*

The recurrent set will be recursively computed by starting with $\mathcal{R}_0(A) = A$ and by finding, at each step $i \geq 1$, the set $\mathcal{R}_i(A)$, which is the set of all $s \in A$ from which there can be enforced at least i revisits to set A. Then, because A is finite, the decreasing sequence $A \supseteq \mathcal{R}_1(A) \supseteq \mathcal{R}_2(A) \ldots$ will reach a stationary value (which can be the empty set) for some $i \leq |A|$, and that value is $\mathcal{R}(A)$. The actual details of computing $\mathcal{R}_i(A)$ will be given after Definition 7.

Definition 6 (Attractor set). *The attractor set of A, denoted by $\mathcal{A}(A)$, is the set of all $s \in S_P$ from which there can be enforced a visit to set A in zero or more steps.*

Enforcing a visit in zero steps is equivalent to starting from A and applying no input. The attractor set will be the stationary value of an increasing sequence: $\mathcal{A}_0(A) \subseteq \mathcal{A}_1(A) \subseteq \mathcal{A}_2(A) \ldots$, where $\mathcal{A}_i(A)$ is the set of all $s \in S_P$ from which there can be enforced a visit to set A in at most i steps. It is easy to see that the recursion starts with $\mathcal{A}_0(A) = A$ and, at each step, $\mathcal{A}_{i+1}(A)$ is computed as the union of $\mathcal{A}_i(A)$ with the set of all $s \in S_P \setminus \mathcal{A}_i(A)$ from which there can be enforced a visit to set $\mathcal{A}_i(A)$ in one step, for $i \geq 0$. The stationary set ($\mathcal{A}_{i+1}(A) = \mathcal{A}_i(A)$) will again be reached in a finite number of steps $i \leq |S_P|$.

The winning region W_P is given by $W_P = \mathcal{A}(\mathcal{R}(F_P))$: once $\mathcal{R}(F_P)$ is reached, we are certain that we can revisit states from F_P infinitely often. However, in order to effectively compute the recurrent set of a given subset of states, we need one more definition.

Definition 7 (Proper attractor). *The proper attractor of A, denoted by $\mathcal{A}^+(A)$, is defined as the set of all $s \in S_P$ from which there can be enforced a visit to set A in one or more steps.*

The proper attractor is computed similarly to the attractor set, but the following differences appear: we start with $\mathcal{A}_0^+(A) = \emptyset$ and, at each iteration, we compute $\mathcal{A}_{i+1}^+(A)$ as the union of $\mathcal{A}_i^+(A)$ with the set of all $s \in S_P \setminus \mathcal{A}_i^+(A)$ from which there can be enforced a visit to set $\mathcal{A}_i(A) \cup A$ in one step. This comes from the fact that the proper attractor set requires at least one step to be taken (one input to be applied) in order to visit set A, while the regular attractor from Definition 6 consider as a visit the situation of starting from set A and taking no transition. Because of this difference, the proper attractor is suitable for computing the recurrent set, by using the recurrence $\mathcal{R}_{i+1}(A) = \mathcal{R}_i(A) \cap \mathcal{A}^+(\mathcal{R}_i(A))$: at each step, keep only states from $\mathcal{R}_i(A)$ from where a revisit to $\mathcal{R}_i(A)$ can be enforced in a strictly positive number of steps.

We now have all the tools for solving the Büchi game on the product automaton P. Due to space constraints, we do not include the corresponding algorithm here, and we refer to the technical report from `http://iasi.bu.edu/~software/nondet.htm`. The idea of solving the Büchi game on P is to first compute $\mathcal{R}(F_P)$ (using the recurrence given after Definition 7), and then, if the resulting set is nonempty, the set $W_P = \mathcal{A}(\mathcal{R}(F_P))$ is computed (as described after Definition 6). The winning strategy π_P is constructed during the computation of these sets, by searching inputs from Σ

guaranteeing the satisfaction of definitions for recurrent and attractor sets, respectively. The obtained solution is the set W_P and the memoryless strategy $\pi_P : W_P \to \Sigma$. If the set $W_{P0} = S_{P0} \cap W_P$ is nonempty, then there exist initial states of P from where F_P will be visited infinitely often. Otherwise, our Büchi game with inputs has no solution, and correspondingly Problem 1 is infeasible. The solution from this section is complete, and its correctness is guaranteed by construction.

5.3 Constructing the Control Strategy for T

If there is a solution for winning the Büchi game on the product automaton P (set W_{P0} is nonempty), we have to adapt this solution to our initial transition system T. First, the set of initial states of T from where the LTL formula can be satisfied is $Q_0 = \alpha(W_{P0})$, where map $\alpha : S_P \to Q$ is just the projection of states from P to Q. Second, the control strategy for T will be an automaton C obtained from the memoryless strategy π_P in the winning region W_P. The input applied to C will be the current state of T, and the output of C will give the next input to be applied to T.

The control automaton C is the tuple $C = (S, Q, s_0, \tau, \pi, \Sigma)$, where:

- S is the set of states of B,
- Q is the input set, equal with the set of states of T,
- s_0 is the initial state of the deterministic Büchi automaton B,
- $\tau : S \times Q \to S$ is the memory update function, $\tau(s, q) = \delta_B(s, q)$ if $(q, s) \in W_P$, and $\tau(s, q)$ undefined otherwise,
- $\pi : S \times Q \to \Sigma$ is the output function, $\pi(s, q) = \pi_P((q, s))$ if $(q, s) \in W_P$, and $\pi(s, q)$ undefined otherwise.

The correctness of the control automaton C can be verified as follows: if we equip C with the set of final states F, then the product automaton $T \times C$ will have the same states as P, its transitions will be the subset of transitions of P that can appear during the winning of a Büchi game, and the strategy for applying inputs is exactly π_P.

To summarize, the solution to Problem 1 is given by the set of initial states $Q_0 = \alpha(W_{P0})$ and the feedback control automaton C. Whenever T starts from an initial state in set Q_0, the satisfaction of the LTL formula corresponding to the Büchi automaton B is guaranteed by the controller C, which, at each step, reads the current state of T, uses map π to determine the next input to be applied to T, and updates its own internal state by using the map τ.

Since the solution from section 5.2 is complete, the overall procedure proposed in this paper for solving Problem 1 is complete. On complexity, the running time of the overall three-step procedure is $O(|Q|^2 \cdot |S|^2 \cdot |\Sigma|)$ (proofs are omitted due to space constraints). If the specification is given as an LTL formula, to this we need to add the running time for the conversion of the formula to a Büchi generator, which is at most double exponential in the length of the formula [25]. If smaller fragments of LTL are considered, then the construction of the Büchi generator can be more efficient. For example, for the LTL fragment which includes the example in Eqn. (4), exponential complexity can be achieved [25]. Moreover, note that this upper bounds for complexity are very rarely attained in practice.

The three-step procedure proposed in this paper has been implemented in Matlab. The user-friendly interface takes as input the transition system T and the Büchi automaton B, and returns the control automaton C. The software package is freely downloadable from `http://iasi.bu.edu/~software/nondet.htm`.

6 Case Study Revisited

Let us now revisit the case study from Section 2, which requires to find initial states and feedback controllers for system (1) such that all trajectories of the corresponding closed loop system satisfy specification (2), *i.e.,* LTL formula (3). The deterministic Büchi automaton corresponding to the formula (not shown due to space constraints) has 4 states, one final state, and 56 transitions, out of which 52 are self transitions.

We start by constructing a transition system T with states $Q = \{q_1, \ldots, q_{14}\}$ corresponding to the partition elements and with transitions capturing the ability of designing affine feedback controllers such that a triangle either becomes an invariant for the closed loop system, or it is left in finite time to one or several neighbors. To compute such controllers, we used the method developed in [16], which consists of polyhedral operations only. We first check transitions to one neighbor. Then, if some neighbor(s) cannot be reached, we check transitions to the possible pairs of neighbors that include the non-reachable one(s). We stop either when every neighbor can be reached (through deterministic or non-deterministic transitions), or when all combinations of exit facets were checked (including the set of all three facets). The resulting transition system (shown in Figure 2 (b)) has 38 transitions, out of which 27 are deterministic.

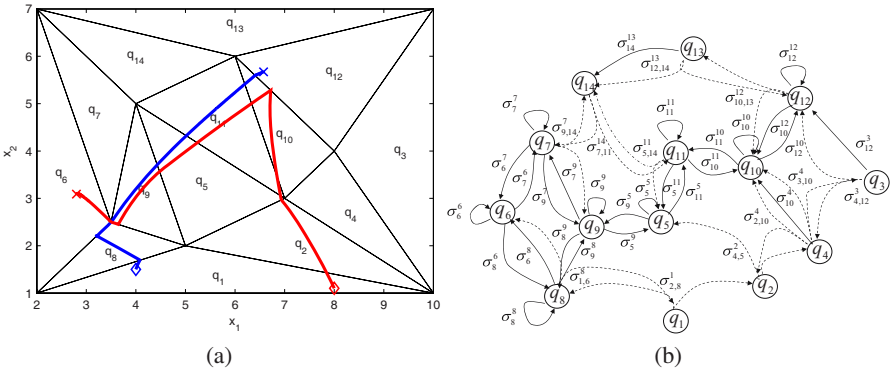

Fig. 2. (a) Two continuous trajectories of the controlled system, starting from the points marked with "$\diamond$" ($x = (4, 1.5)$ and $x = (8, 1.1)$, respectively) and asymptotically converging to the points marked with "X". (b) The nondeterministic transition system modelling feedback controllers making a triangle an invariant and driving all states to one or several neighbors. Deterministic transitions are shown in solid line, while non-deterministic transitions are shown in dashed line. $\sigma^i_{j,k}$ labels a controller driving all initial states in q_i to q_j or q_k.

By using the three-step approach from Sect. 5, we conclude that formula ϕ can be satisfied by trajectories starting from any triangle q_i, $i = 1, \ldots, 14$. The product

automaton has 56 states, which appear all in the winning region W_P. The control automaton C has 4 states. The computation for all three steps took less than one second. We skip the exact details on how this discrete control strategy is applied to the continuous system (1). Roughly, a discrete transition in T takes place when the current triangle is left. Each input to be applied to T is mapped to an affine feedback controller, and applied as long as the continuous trajectory evolves in the current triangle.

In Fig. 2 (a), we show two continuous trajectories starting in region q_1 and corresponding to the strategy imposed by the control automaton C. Even though the continuous trajectories reach different sequences of triangles, they both satisfy the formula. Each trajectory converges to a point marked by "X", inside q_6 and q_{12}, respectively. Note that the solution presented here is less conservative than the one shown in Section 2, which was based on control-to-facet problems and deterministic transition systems. We also solved the (discrete part of the) problem by using two other (conservative) approaches: (1) direct translation to an LTL game on a graph (see Sect. 4), and (2) LTL game played on an augmented transition system (see [21]). Notably, as in Section 2, these two methods returned that the formula can be satisfied by starting from any triangle except q_1, q_2, q_{13}, q_{14}.

7 Conclusion

We developed a method for control of a nondeterministic transition system from a specification given as a temporal logic formula generated by a deterministic Büchi automaton. The method is complete and scales polynomially with the size of the Büchi generator. We illustrated the application of the method to the control of a continuous planar affine system from a specification given as an LTL formula over regions in a triangulated environment.

The method proposed here is quite general, and can be used whenever a finite transition system representation of a control problem can be constructed (*e.g.,* multi-affine dynamics and rectangular partitions). Therefore, it provides the first steps towards the construction of expressive specification languages for symbolic control. An immediate application is automatic planning and control of robot motion, where triangulation and rectangular grids are the most used partitioning schemes, and task specifications are naturally given as temporal and logic statements about the reachability of regions of interest in the robot environment.

References

1. Milner, R.: Communication and concurrency. Prentice-Hall, Englewood CliDs, NJ (1989)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) HS 1991 and HS 1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993)
4. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What is decidable about hybrid automata? J. Comput. Syst. Sci. 57, 94–124 (1998)

5. Lafferriere, G., Pappas, G.J., Sastry, S.: O-minimal hybrid systems. Math. Control, Signals, Syst 13(1), 1–21 (2000)
6. Tabuada, P., Pappas, G.J.: Linear time logic control of discrete-time linear systems. IEEE Transactions on Automatic Control 51(12), 1862–1877 (2006)
7. Tabuada, P.: Symbolic control of linear systems based on symbolic subsystems. IEEE Transactions on Automatic Control 51(6), 1003–1013 (2006)
8. Girard, A.: Approximately bisimilar finite abstractions of stable linear systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 231–244. Springer, Heidelberg (2007)
9. Alur, R., Dang, T., Ivancic, F.: Reachability analysis of hybrid systems via predicate abstraction. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, Springer, Heidelberg (2002)
10. Tiwari, A., Khanna, G.: Series of abstractions for hybrid automata. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, Springer, Heidelberg (2002)
11. Belta, C., Habets, L.: Constructing decidable hybrid systems with velocity bounds. In: 43rd IEEE Conference on Decision and Control, Paradise Island, Bahamas (2004)
12. Habets, L., van Schuppen, J.: A control problem for affine dynamical systems on a full-dimensional polytope. Automatica 40, 21–35 (2004)
13. Belta, C., Habets, L.: Control of a class of nonlinear systems on rectangles. IEEE Transactions on Automatic Control 51(11), 1749–1759 (2006)
14. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
15. Kloetzer, M., Belta, C.: A fully automated framework for control of linear systems from LTL specifications. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 333–347. Springer, Heidelberg (2006)
16. Habets, L., Collins, P., van Schuppen, J.: Reachability and control synthesis for piecewise-affine hybrid systems on simplices. IEEE Trans. Aut. Control 51, 938–948 (2006)
17. Kloetzer, M., Habets, L., Belta, C.: Control of rectangular multi-affine hybrid systems. In: 45th IEEE Conference on Decision and Control, San Diego, CA (2006)
18. Thomas, W.: Infinite games and verification. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 58–64. Springer, Heidelberg (2002)
19. Wallmeier, N., Hütten, P., Thomas, W.: Symbolic synthesis of finite-state controllers for request-response specifications. In: H. Ibarra, O., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 113–127. Springer, Heidelberg (2003)
20. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005)
21. Kloetzer, M., Belta, C.: Managing non-determinism in symbolic robot motion planning and control. In: IEEE International Conference on Robotics and Automation, Rome, Italy (2007)
22. Wolper, P., Vardi, M., Sistla, A.: Reasoning about infinite computation paths. In: Nagel, E., et al. (eds.) Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, Tucson, AZ, pp. 185–194 (1983)
23. Safra, S.: Complexity of automata on infinite objects. PhD thesis, The Weizman Institute of Science, Rehovot, Israel (1989)
24. Fainekos, G.E., Loizou, S.G., Pappas, G.J.: Translating temporal logic to controller specifications. In: 45th IEEE Conference on Decision and Control, San Diego, CA (2006)
25. Alur, R., Torre, S.L.: Deterministic generators and games for LTL fragments. In: 16th IEEE Symposium on Logic in Computer Science, LICS 2001, pp. 291–300 (2001)
26. Kumar, R., Garg, V.K.: Modeling and Control of Logical Discrete Event Systems. Kluwer, Boston, MA (1995)