2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Managing non-determinism in symbolic robot motion planning and 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— **We study the problem of designing control strategies for non-deterministic transitions systems enforcing the satisfaction of Linear Temporal Logic (LTL) formulas over their set of states. We focus on finite transition systems with inputs, which are often encountered when solving motion planning problems by using discrete quotients induced by a given partition of the state space. Our approach solves the problem conservatively using LTL games, and consists of the following three steps: (1) the original transition system is transformed into a transition system on which an LTL game can be played, (2) a solution of the LTL game on the new transition system is obtained, and (3) an interface between this solution and the initial transition system is constructed. The correctness of the method is ensured by design. The advantages and conservativeness of our approach are discussed and illustrated by simple examples.**

I. INTRODUCTION

Motion planning and control of robots with nontrivial dynamics or kinematics is usually a two step process. In the first step, a (cellular) decomposition of the C-space is constructed, and a "discrete" path is generated by a search in the quotient graph [1], [2]. In the second step, a reference trajectory compatible with the robot dynamics or kinematic is generated, and robot control laws are designed to follow the trajectory. A very attractive alternative to this is simultaneous planning and control, in which the generation of the discrete solution in the partition quotient is performed at the same time with the assignment of vector fields in the regions of the partition (robot control laws), while taking into account the restrictions imposed by robot under-actuation and speed constraints [3], [4], [5], [6]. In addition, enrichment of the specification language from the classical planning task "go from A to B" to temporal logic specifications (*e.g.,* "visit either A or B", "reach A and then B infinitely often") leads to symbolic approaches to simultaneous planning and control, where discrete abstractions are used to provide a formal link between the continuous robot control system and the discrete representation of the environment, and algorithms resembling model checking are used to provide a solution to the discrete problem [7], [8].

All the approaches enumerated above for simultaneous planning and control face a common problem: the restrictions imposed by the robot dynamics or kinematics can, in general, lead to non-deterministic representations of the discrete problem. For example, the "prepares" relationship between a collection of policies in [4] results in non-determinism of the discrete abstraction. Recent results in control of affine systems in simplices [9] and of multi-affine systems in rectangles [10] show that even though controllers determining the transition of a robot to a specific neighbor might fail to exist, controllers guaranteeing the transition to a set of neighbors can be found. In the transition system corresponding to the discrete part of the problem, this means non-determinism.

Motivated by the above, in this paper we focus on the discrete part of simultaneous motion planning and control, and consider the following problem: given a non-deterministic transition system with inputs, and a Linear Temporal Logic (LTL) formula over its set of states, determine a set of initial states and a control strategy so that the produced trajectory satisfies the formula. This problem is quite general, and, to the best of our knowledge, there is no available computational framework providing a solution. However, it is related to LTL games [11]. Also, it is possible that solutions can be found by using results from the control of discrete event systems from specifications given as ω-regular expressions (languages) over inputs [12] or by using tree automata on infinite objects [13].

In this work, we advocate the use of LTL games [11] for the construction of a (conservative) solution to the problem. Our fully automatic computational framework consists of three main steps. First, we convert the original transition system into a form in which an LTL game can be played. Second, we find a solution to the LTL game in the form of a feedback automaton. Third, the solution of the game is used to generate a control strategy for the initial transition system. We illustrate our approach for the case of a triangulated planar environment, where the assignment of affine feedback controllers in the triangles using the method developed in [9] leads to non-determinism.

The remainder of the paper is organized as follows. Section II provides some definitions necessary throughout the paper. The problem is formulated in Section III and our solution is presented in Section IV. Simulation results are given in Section V and we conclude with final remarks in Section VI.

1-4244-0602-1/07/$20.00 ©2007 IEEE.

II. PRELIMINARIES

For a finite set A, we use $|A|$, 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.

A. Transition Systems and Linear Temporal Logic

Definition 1: [Non-deterministic transition system] A finite non-deterministic transition system is a tuple $T = (Q, \Sigma, \delta, h, O)$, where:

- Q is a finite set of states,
- Σ is a finite input alphabet,
- $\delta : Q \times \Sigma \to 2^Q$ is a transition function,
- $h : Q \to O$ is the observation map,
- O is the set of observables.

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 $r \in Q^\omega$, $r = r_1r_2r_3 \ldots$ with the property that $r_1 = q$ and $\forall i \geq 1$, $r_{i+1} \in \delta(r_i, \sigma_i)$. A trajectory r of T produces a *word* $w \in O^\omega$ defined as $w = w_1w_2w_3 \ldots$, $w_i = h(q_i)$, for all $i \geq 1$.

A transition system $T = (Q, \Sigma, \delta, h, O)$ for which the observation map is identity (*i.e.,* the states are of interest $Q = O$, and can be observed) is denoted for simplicity by $T = (Q, \Sigma, \delta)$.

Definition 2: [Syntax of LTL formulas] An LTL formula over O is recursively defined as follows:

- Every observable $o \in O$ is a formula, and
- If ϕ_1 and ϕ_2 are formulas, then $\neg\phi_1$, $\phi_1 \vee \phi_2$, $\bigcirc\phi_1$, $\phi_1\mathcal{U}\phi_2$ are also formulas.

The semantics of LTL formulas are given over words of transition system T.

Definition 3: [Semantics of LTL formulas] The satisfaction of formula ϕ at position $i \in \mathbb{N}$ of word w, denoted by $w_i \vDash \phi$, is defined recursively as follows:

- $w_i \vDash o$ if $o = w_i$,
- $w_i \vDash \neg\phi$ if $w_i \nvDash \phi$,
- $w_i \vDash \phi_1 \vee \phi_2$ if $w_i \vDash \phi_1$ or $w_i \vDash \phi_2$,
- $w_i \vDash \bigcirc\phi$ if $w_{i+1} \vDash \phi$,
- $w_i \vDash \phi_1\mathcal{U}\phi_2$ if there exist a $j \geq i$ such that $w_j \vDash \phi_2$ and for all $i \leq k < j$ we have $w_k \vDash \phi_1$

A word w satisfies an LTL formula ϕ, written as $w \vDash \phi$, if $w_1 \vDash \phi$.

The symbols $\neg$ and $\vee$ stand for negation and disjunction. The Boolean constants $\top$ and $\perp$ are defined as $\top = \pi \vee \neg\pi$ and $\perp = \neg\top$. The other Boolean connectors $\wedge$ (conjunction), $\Rightarrow$ (implication), and $\Leftrightarrow$ (equivalence) are defined from $\neg$ and $\vee$ in the usual way. The unary *temporal operator* $\bigcirc$ is called *next*, and the binary *temporal operator* $\mathcal{U}$ is called the *until* operator. Two useful additional temporal operators, "eventually" and "always" can be defined as $\Diamond\phi = \top\mathcal{U}\phi$ and $\Box\phi = \phi\mathcal{U}\perp$, respectively. Formula $\Diamond\phi$ means that ϕ becomes eventually true, whereas $\Box\phi$ indicates that ϕ is true at all positions of a trajectory.

Remark 1: In general, the semantics of LTL formulas are given over infinite words in the power set of a set of atomic propositions. We use the simplified semantics of Definition 3 motivated by the problem formulated in Section III.

A Büchi automaton is a finite state automaton accepting infinite strings. The definition of a Büchi automaton and its acceptance condition is beyond the scope of this paper, and we refer the interested reader to [13]. For any LTL formula, there exists a non-deterministic Büchi automaton accepting all and only the runs satisfying it [14] (this automaton is also called a generator of the LTL formula).

B. LTL Games

An LTL game is defined on a graph $G = (V, E, h, O)$, where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, O is a set of observables, and $h : V \to O$ is an observation map. The game specification is an LTL formula over the observable set O [15] (see also [11] for the case of graphs with no observables). There are two players in the game: $\mathcal{P}$ (the player) and $\mathcal{A}$ (the adversary). The set of vertices V is partitioned in two sets: V_p, which is the set of states from which the player can choose a move (an edge leading to the next vertex), and V_a, which is the set of states from which the adversary has a choice of an edge. We assume that the current vertex from set V is always known, and not just its observable.

An (infinite) play consists of an infinite sequence of states resulted from an infinite sequence of transitions (edges) chosen by the two players. The player $\mathcal{P}$ wins a play if the produced word $w \in O^\omega$ satisfies the LTL formula. Starting from a given initial state, the player has a winning strategy if, whenever the current state is in V_p, she manages to choose edges such that she wins the current play, no matter what edges the adversary chooses when the current state is in V_a. The goal of an LTL game is to find a set of initial states $W_s \subseteq V$ from where the player has winning strategies and a winning strategy for plays starting in those initial states.

The standard algorithm for solving an LTL game involves the transformation of the non-deterministic Büchi automaton corresponding to the LTL formula into a deterministic generator [16]. Motivated by the simplicity of exposition and by some complexity issues, we assume that the LTL formulas we are dealing with accept deterministic Büchi generators (there are cases when this is not true [17], and algorithms for solving LTL games become more complicated). For more details of the steps involved in solving LTL games the interested reader is referred to [16], [15].

The result of solving an LTL game on $G = (V, E, h, O)$ (as outlined above) is a set of initial states $W_s \subseteq V$ (if non-empty) and a winning strategy W, which is a strategy (as in Definition 4) guaranteeing the satisfaction of the LTL formula whenever the initial state is in the set W_s.

Definition 4: A strategy is an automaton $W = (S, V, O, s_0, \tau, \pi)$, where:

- S is the finite set of states (the memory),
- V is the input alphabet,
- O is the set over which the LTL formula ϕ was defined,

- $s_0 \in S$ is the initial state (initial memory),
- $\tau : S \times O \rightarrow S$ is the memory update function,
- $\pi : S \times V_p \rightarrow V$ is the function giving the chosen move when the current state of G is in V_p.

The states S result from the states of the deterministic automaton (*e.g.,* Büchi) corresponding to the LTL formula. The current vertex of G gives the input of W and it is used for generating the player's moves, while the corresponding observable is used for updating the internal memory of W. Thus, given the current state (memory) of W, the strategy π depends on the current state of G, while the memory update function τ depends only on the observable of this state.

The relation between graphs and transition systems is immediate. When playing an LTL game on a transition system, instead of choosing the next state, we choose an input producing the desired transition. This is possible if player's states have only deterministic outgoing transitions, and then the function π from the winning strategy will take values in the input set of the transition system instead of V.

III. PROBLEM FORMULATION AND APPROACH

In this paper we consider the following problem:

Problem 1: Given a non-deterministic transition system $T = (Q, \Sigma, \delta)$ and an LTL formula ϕ over Q, find a set of feasible initial states $Q_0 \subseteq Q$ and a control strategy such that formula ϕ is satisfied by all resulting infinite words.

Remark 2: If the transition system T was deterministic (*i.e.*, $\delta : Q \times \Sigma \rightarrow Q$), then a solution to Problem 1 could be found by using an idea similar to model checking [18], [7], [19]: the LTL formula ϕ is transformed into a Büchi automaton B_ϕ and the product automaton $T \times B_\phi$ is computed. 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.

We propose to use the framework of LTL games to provide a solution to Problem 1. For this, we need to reformulate the problem from a graph to a transition system, and then define a partition of the set of states into player's states and adversary's states.

Remark 3: One quick way of solving this would be to label as adversary's state each state from where there exists at least one input yielding a non-determinist transition. However, such an approach would be pretty conservative, because if such an adversary's state had more than one feasible input, we wouldn't use our freedom of choosing one from these inputs and thus restricting the set of possible next states.

Our approach involves three main steps. First, we create a new transition system T_g with observables, where we choose the player's states by looking for states and inputs with deterministic behavior. T_g will, in general, have more states and inputs than T, and its observable set will be Q. Second, we search for a winning strategy for the LTL game on T_g. Finally, when such a strategy exists, we adapt it to a control strategy providing at each step the input to be applied to our initial transition system T.

IV. CONTROL STRATEGY USING LTL GAMES

We start by transforming T into another transition system $T_g = (Q_g, \Sigma_g, \delta_g, h_g, Q)$, where:

- Q_g is the finite set of states, partitioned into sets Q_p (player's states) and Q_a (adversary's states),
- Σ_g is the new alphabet, $\Sigma \subseteq \Sigma_g$,
- $\delta_g : Q_g \times \Sigma_g \rightarrow 2^{Q_g}$ is the transition function,
- $h_g : Q_g \rightarrow Q$ is the observation map.

The construction of T_g is described in Section IV-A. Here we give some definitions and outline how the game is played. The transition function δ_g is defined as:

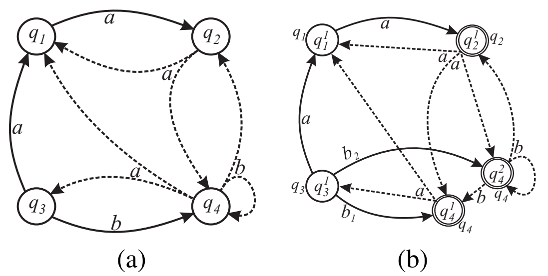
- its restriction to $Q_p \times \Sigma_g$ is deterministic, *i.e.* $\forall q \in Q_p$ and $\forall \sigma \in \Sigma_g$, $|\delta_g(q, \sigma)| \leq 1$,
- its restriction to $Q_a \times \Sigma_g$ is non-deterministic, and there is only one feasible (enabled) input in every state from Q_a, *i.e.* $\forall q \in Q_a$, there exists an unique input $\sigma \in \Sigma_g$ such that $|\delta_g(q, \sigma)| \geq 2$ and all other inputs $\sigma' \in \Sigma_g \setminus \{\sigma\}$ are not feasible at q (or $|\delta_g(q, \sigma')| = 0$).

These properties of δ_g show that when the current state is in Q_p, the player $\mathcal{P}$ can always choose the next state from a feasible set (by applying a feasible input), and when the current state is in Q_a, $\mathcal{P}$ has no control on the next state (we only know that the next state will be in a subset of Q_g, given by the unique feasible input). Thus, the structure of T_g is appropriate for an LTL game as described in Section II-B. By solving the LTL game over T_g with winning condition ϕ, one can obtain a set of winning initial states, $W_s \subseteq Q_g$, and, if $W_s \neq \emptyset$, a winning strategy in the form of a finite automaton, similar to the one from Definition 4. When W_s is nonempty, the winning strategy is $W = (S, Q_g, Q, s_0, \tau, \pi)$, where:

- S is a finite set of states,
- Q_g is the input alphabet,
- Q is the set over which the LTL formula ϕ was defined,
- $s_0 \in S$ is the initial state,
- $\tau : S \times Q \rightarrow S$ is the memory update function,
- $\pi : S \times Q_p \rightarrow \Sigma_g$ is the function giving the choice of the next applied input (move) when the current state of T_g is in the set Q_p.

The difference between this winning strategy and the one from Definition 4 is that here π takes values in Σ_g instead of Q_g. Because of the mentioned properties of δ_g, every move the player should make when the play is in Q_p corresponds to an unique input from Σ_g. In order to obtain an input to be applied to T_g in every state, the map π is extended to $S \times Q_g$, by letting $\pi(s, q) = \sigma$, where $\sigma \in \Sigma_g$ is the only feasible input in state q, $\forall q \in Q_a$.

The winning strategy given by automaton W is implemented in a feedback form. At each step, the following actions are performed (a step is the interval between two successive transitions of T_g): (1) it reads the current state of T_g, (2) according to function π, it applies an input from the set Σ_g to T_g, and (3) it updates its state according to function τ. If the initial state of T_g is in the set W_s, this feedback control strategy guarantees that the game will be won. Note



(a) (b)

Fig. 1. (a) Transition system T: q_1 and q_3 are *det* states, q_2 is *ndet*, and q_4 is *mix*. (b) Obtained T_g: q_4 from T was split in two *ndet* states in T_g and two new inputs appeared in Σ_g. In T_g, the observables are placed near the states, and the adversary's states are doubly encircled. All deterministic transitions are represented with continuous lines, non-deterministic ones with dashed lines, and each input is placed near the corresponding transition(s).

that this feedback controller assumes that we can always read the current state of T_g and not only its observable.

A. Construction of T_g

We present here the main ideas used for transforming the initial transition system T into the new transition system T_g. Due to space constraints, we do not include any algorithm and we refer to [20] for full pseudo-codes.

For creating the states and the observation map of T_g, we label the states from Q by a function $l : Q \to \{det, ndet, mix\}$: a state with only deterministic outgoing transitions or without outgoing transitions is labelled with *det*, a state with only one feasible input, for which the outgoing transitions are non-deterministic, is labelled with *ndet*, and a state with more feasible inputs, from which at least one yields non-deterministic transitions, is labelled with *mix*. Each *mix* state is split in one or more *ndet* states and at most one *det* state in the new transition system T_g (for simplicity of writing, we assume that the attributes *ndet* and *det* for states of T_g have exactly the same meaning as in T). The observable of all these new states is the old *mix* state. Each *det* (respectively *ndet*) state from T corresponds to a *det* (respectively *ndet*) state in T_g. Thus, T_g will have only *det* (in the subset Q_p) and *ndet* states (in the subset Q_a). For a better understanding we present in Fig. 1 a simple example of transforming a transition system T into T_g.

The input alphabet Σ_g and the transition function δ_g are created after the set of states $Q_g = Q_p \cup Q_a$ is obtained. T_g has a larger input set than T because the *det* states from T must remain *det* in T_g. This can be explained by following the example in Fig. 1: in T, the *det* state q_3 has a transition with input b to the *mix* state q_4. q_3 corresponds to one state of T_g (q_3^1), but q_4 is split in two states (q_4^1 and q_4^2). Since we need deterministic transitions for going from q_3^1 to either q_4^1 or q_4^2, we introduce the new inputs b_1 and b_2. Thus, by applying b_1 when T_g is in q_3^1, the next state is q_4^1 and there is only one next feasible input (a). In T this has the effect of applying input b while in q_3 and enforcing input a when q_4 is reached. Similarly, when b_2 is applied to T_g while in q_3^1, this induces the sequence of inputs b, b starting from q_3 in T. Although the adversary has full control on the transitions from q_4^1 and q_4^2 in T_g, the player has control in going to either q_4^1 or q_4^2, thus restricting the adversary's power. Following the same idea, each *det* state of T_g (including those obtained by splitting *mix* states of T) might add some new inputs to the alphabet Σ_g. In order to adapt these new inputs to inputs of T, a map $\alpha : \Sigma_g \to \Sigma$ is created, which will be used by the controller for T. For all inputs in set Σ, α is just the identity map.

The *ndet* states of T_g (from adversary's set Q_a) will not add new inputs. Each of these states will have only one feasible input (from set Σ), and thus the adversary power in these states comes only from the non-determinism induced by that feasible input.

B. Solution to Problem 1

We now have all the necessary information to present the solution to Problem 1. The set of initial states of T from where the controller guarantees the satisfaction of the LTL formula ϕ is:

$$Q_0 = \{q \in Q \mid \exists q' \in W_s \text{ s.t. } h_g(q') = q\} \tag{1}$$

The set Q_0 from (1) is in accordance with the solution of the LTL game for the transition system T_g: if the initial state of T_g is in the set W_s, then the strategy generated by automaton W guarantees the satisfaction of formula ϕ by any produced run of T_g. Of course, $Q_0 = \emptyset$ if and only if $W_s = \emptyset$, case when we conclude that we cannot solve Problem 1.

If $Q_0 \neq \emptyset$, the control strategy for the transition system T will be a feedback controller, which will also include the interface between T and the play on T_g, supervised by W. Again, we briefly describe the operations performed by this controller and we refer to [20] for an algorithm accompanied by detailed explanations.

At each discrete step, a correct state of T_g is first picked: this state must be in concordance with the previous state of T_g and the previously applied input, and its observable must equal the current state of T. Then, based on the winning strategy W, an input for T_g is found, which is then adapted to an input for T by using map α. Whenever the resulted input for T_g is in the set $\Sigma_g \setminus \Sigma$, the adversary's power in the next state of T will be reduced (for example, see the case depicted in Fig. 1, when input b_1 or b_2 appears).

For a better understanding, we complete the example from Fig. 1 by considering the formula $\phi = \diamondsuit(q_4 \wedge \diamondsuit q_1)$, meaning "visit q_4 and then q_1". By using the presented approach, we obtain $Q_0 = \{q_3, q_4\}$. When T starts from q_3, the first two inputs applied by the controller are b, a (corresponding to input b_1 of T_g). If the resulted state is q_3, the input a will be applied and q_1 is reached. After q_1 is visited, the controller will keep applying any of the feasible inputs in the visited states (in our implementation, a will be always applied). Note that by trying to solve the same problem by using the approach from either Remark 2 or Remark 3, we would get no solution for the problem (in the first case there are no deterministic transitions out of q_4, and in the second case, once q_4 is reached, the adversary can keep choosing the self-loop in this state).

C. Conservativeness and Complexity

Our approach is conservative in the following sense: when creating the transitions of T_g from *ndet* (adversary's) states, we don't restrict the set of inputs to be applied to T at the next discrete step. For illustrating this, consider the example in Fig. 1 and the formula $\phi = \Diamond q_1$. Our approach gives $Q_0 = \{q_1, q_3, q_4\}$, while one can observe that ϕ can be also satisfied by starting from q_2 and keep applying input a, which eventually leads to q_1 being visited. However, approaches from Remarks 2 and 3 give only strategies from $Q_0 = \{q_1, q_3\}$. This problem does not appear in the case of *det* states of T_g, because of the inputs in $\{\Sigma_g \setminus \Sigma\}$, which restricts the set of choices for the next input. If there are no transitions between *mix* and *ndet* states of T, then this source of conservativeness does not appear.

Another source of conservativeness in our approach is that we assumed an LTL formula over the states of T. A more general setting would involve a transition system T with an observation map and an LTL formula over the set of observables. If states of T having the same observable were indistinguishable (not like in the case of T_g), the proposed method wouldn't be suitable, because we couldn't find a state of T_g (with the correct observable) and be sure that the feasible inputs in that state are also feasible in the real state of T. However, this formulation is motivated by our particular motion planning application, where we assume that the environment is partitioned and the resulting regions are labelled (see example in Section V).

Finally, when our approach fails in providing a solution, we declare Problem 1 to be unfeasible. Other approaches could use an iterative procedure obtaining at each step a finer discrete representation (tailored to a specific problem) and trying to solve Problem 1 for that refined transition system.

The upper bound complexity of the presented approach is in general high, because an LTL game on a graph T_g is solvable in polynomial time in the size of the graph and in double-exponential time in the size of the LTL formula [21], [22]. However, there have been isolated some LTL fragments accepting deterministic Büchi automata and guaranteeing a much lower complexity [22].

V. CASE STUDY

To illustrate the approach proposed in the paper, we consider a simple planning and control problem for a planar continuous system evolving in a rectangular triangulated environment (see Fig. 2). The dynamics, rectangular bounds, and control constraints are given in equation (2). We assume that the environment is partitioned in 8 triangles (labelled by q_i, $i = 1, \ldots, 8$ in Fig. 2), and the task is specified by the LTL formula $\phi = \Diamond q_4 \wedge \Diamond q_7$. In other words, the task requires that regions q_4 and q_7 be visited (in any order) by the trajectories of the closed loop system.

$$\dot{x} = \begin{bmatrix} -0.3 & 0.4 \\ 0.2 & -0.5 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}, \quad (2)$$
$$x \in [-3, 4] \times [-3, 3], \; u \in [-1, 1] \times [-1, 1]$$

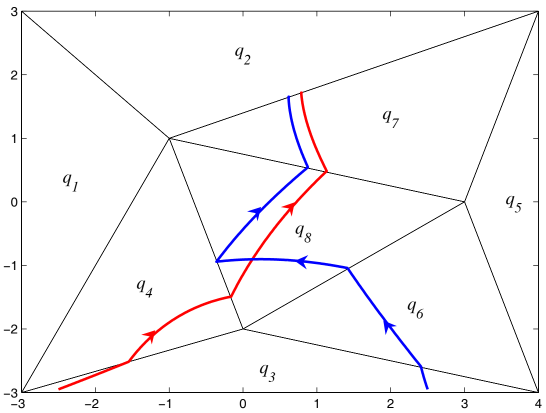


Fig. 2. Triangular partition (thin black lines) and two continuous trajectories starting from region q_3 and satisfying formula ϕ.

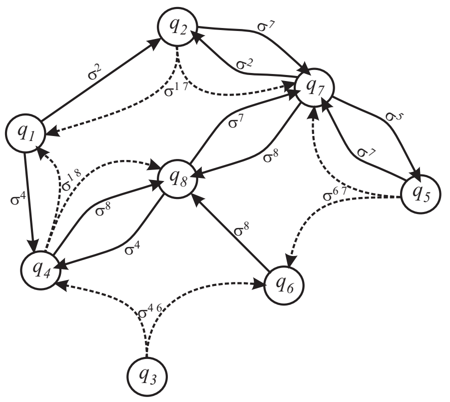


Fig. 3. Transition system T corresponding to the partition in Fig. 2. Deterministic transitions are represented with continuous lines, non-deterministic ones with dashed lines, and each input is placed near the corresponding transition(s).

A formal definition of the semantics of the formula over continuous trajectories of the closed loop system is beyond the scope of this paper, and can be found in [19]. However, intuitively, a trajectory satisfies the formula if the word produced by the sequence of regions visited by the system as time evolves satisfies the formula. As in [7], [19], we start by constructing a transition system T with states $Q = \{q_1, \ldots, q_8\}$ corresponding to the partition elements and with transitions capturing the ability of designing affine feedback controllers such that a triangle is either left in finite time to a neighbor, or becomes an invariant for the closed loop system. The construction of such controllers is computationally efficient, since it can be reduced to operations on polyhedral sets [19].

However, unlike in [7], [19], we don't restrict our attention to controllers making a triangle an invariant or driving all initial states in a triangle to one neighbor (this automatically induces a deterministic quotient, with transitions labelled by the corresponding controllers). Enabled by recent results on control of affine systems in simplices [9], we exhaustively check for existence of controllers making a triangle an invariant, driving all initial states in a triangle to a neighbor, two neighbors, and three neighbors. While dramatically reducing the conservativeness of the approach, it induces a non-deterministic transition system. The obtained transition system T is shown in Fig. 3, where the labels (inputs) stand for affine feedback controllers with an obvious meaning. Note that, motivated by the particular form of the formula (which is a reachability type formula) we do not consider the controllers corresponding to invariance, and, therefore, T does not have any self transitions.

By using the approach from Section IV, we conclude that the formula ϕ can be satisfied by trajectories starting from any triangle q_i, $i = 1, \ldots, 8$. The LTL formula ϕ accepts a deterministic Büchi generator B with 4 states, and the constructed transition system T_g has 11 states. The winning strategy W has 4 states and it is automatically generated, together with the set Q_0, after providing the inputs T and B. When implementing the (discrete) controller to the continuous system [19], a discrete transition in T takes place when the current triangle is left, and each input to be applied to T is mapped to an affine feedback controller as described in [9], applied as long as the continuous trajectory evolves in the current triangle.

The most interesting case is that of trajectories initiating in triangle q_3, because its corresponding discrete state has only non-deterministic outgoing transitions. We show two continuous trajectories starting in region q_3 and corresponding to the winning strategy we found as in Section IV. Even though the continuous trajectories reach different sequences of triangles, they both satisfy the formula. Only the first part of each trajectory is shown (because the trajectories are infinite); after region q_7 is hit, the trajectories will keep oscillating between triangles q_7 and q_2.

Solving the same problem by ignoring the non-deterministic transitions of T and using the approach from Remark 2, one would obtain that the formula could be satisfied by starting from any triangle except q_3 (note that state q_3 doesn't have deterministic outgoing transitions in T). By using the method from Remark 3, the adversary would have control on all the outgoing transitions from states q_2, q_3, q_4, q_5, and winning strategies exist only when starting from states q_6, q_7, q_8. In the later case, q_7 will be first visited, and then q_4 (if q_4 was visited first, a smart adversary would keep the play between states q_4 and q_3, and thus the game would be lost).

VI. CONCLUSIONS

In this paper, we developed a control strategy for a non-deterministic transition from a specification given as an LTL formula over its set of states. The method is based on LTL games, and consists of three steps: the construction of a new transition system, the solution of an LTL game, and the generation of the control strategy for the initial system. The problem is motivated by and applied to planning and control of continuous systems from symbolic specifications given as temporal logic formulas.

VII. ACKNOWLEDGMENTS

This work was partially supported by NSF CAREER 0611926 and NSF 0611925 at Boston University.

The authors wish to thank Paulo Tabuada for useful discussions on this topic.

REFERENCES

[1] J. C. Latombe, *Robot Motion Planning*. Kluger Academic Pub., 1991.

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, 2005.

[3] S. Lindemann and S. LaValle, "Computing smooth feedback plans over cylindrical algebraic decompositions," in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2006.

[4] D. Conner, H. Choset, and A. Rizzi, "Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies," in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2006.

[5] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot planning and control in polygonal environments," *IEEE Trans. on Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[6] M. Kloetzer and C. Belta, "A framework for automatic deployment of robots in 2D and 3D environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[7] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: a temporal logic approach," in *Proceedings of the 2005 IEEE Conference on Decision and Control*, Seville, Spain, December 2005.

[8] M. Kloetzer and C. Belta, "Hierarchical abstractions for robotic swarms," in *IEEE International Conference on Robotics and Automation*, Orlando, FL, 2006.

[9] L. Habets, P. Collins, and J. van Schuppen, "Reachability and control synthesis for piecewise-affine hybrid systems on simplices," *IEEE Trans. Aut. Control*, vol. 51, pp. 938–948, 2006.

[10] L. Habets, M. Kloetzer, and C. Belta, "Control of rectangular multi-affine hybrid systems," in *45th IEEE Conference on Decision and Control*, San Diego, CA, 2006.

[11] W. Thomas, "Infinite games and verification," in *Proceedings of the International Conference on Computer Aided Verification, CAV'02*, 2002, pp. 58–64.

[12] R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems*. Boston, MA: Kluwer, 1995.

[13] W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, J. van Leeuwen, Ed. Amsterdam: Elsevier, 1990, pp. 133–191.

[14] 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, p. 112.

[15] S. L. Torre, "Deciding games in LTL fragments," Seminar - Centre Fédéré en Vérification, Belgium, March 2004.

[16] W. Thomas, "Automata theoretic foundations of infinite games," Spring School on Infinite Games and Their Applications, Bonn, March 2005.

[17] S. Safra, "Complexity of automata on infinite objects," Ph.D. dissertation, The Weizman Institute of Science, Rehovot, Israel, 1989.

[18] G. Holzmann, *The Spin Model Checker, Primer and Reference Manual*. Reading, Massachusetts: Addison-Wesley, 2004.

[19] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from LTL specifications," in *The 9th International Workshop on Hybrid Systems: Computation and Control*, 2006, pp. 333–347.

[20] ——, "Managing non-determinism in symbolic robot motion planning and control," Boston University, Tech. Rep., 2006, http//iasi.bu.edu/∼techrep/LTL-nondet.pdf.

[21] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*, 1989, pp. 179–190.

[22] R. Alur and S. L. Torre, "Deterministic generators and games for LTL fragments," in *16th IEEE Symposium on Logic in Computer Science, LICS'01*, 2001, pp. 291–300.