Receding Horizon Temporal Logic Planning for Dynamical Systems

Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray

Abstract— **This paper bridges the advances in computer science and control to allow automatic synthesis of control strategies for complex dynamical systems which are guaranteed, by construction, to satisfy the desired properties even in the presence of adversary. The desired properties are expressed in the language of temporal logic. With its expressive power, a wider class of properties than safety and stability can be specified. The resulting system consists of a discrete planner that plans, in the abstracted discrete domain, a set of transitions of the system to ensure the correct behaviors and a continuous controller that continuously implements the plan. To address the computational difficulties in the synthesis of a discrete planner, we present a receding horizon based scheme for executing finite state automata that essentially reduces the synthesis problem to a set of smaller problems.**

I. INTRODUCTION

Recent advances in computer science, such as the development of a polynomial-time algorithm to construct finite state automata from their temporal logic specifications [1], enable automatic synthesis of digital designs that satisfy a large class of properties even in the presence of an adversary (typically arising from changes in the environments). On the other hand, recent advances in control and the abundance of computational resources enable automatic synthesis of continuous controllers that ensure safety and stability even in the presence of disturbances and modeling errors [2], [3], [4]. In many applications, systems need to perform complex tasks and interact with (potentially adversarial) environments. Such systems usually contain both continuous (physical) and discrete (computational) components. A major challenge is to integrate the methods from computer science and control such that automatic synthesis of such systems is possible.

Hybrid system theory has been developed to handle systems that contain both discrete and continuous components. Control of hybrid systems has been studied extensively but properties of interest are typically limited to stability and safety [5], [6]. For systems to perform complex tasks, a wider class of properties such as guarantee (e.g. eventually perform task 1 or task 2 or task 3) and response (e.g. if the system fails, then eventually perform task 1 or perform tasks 1, 2 and 3 infinitely often in any order) need to be considered. Temporal logics have therefore garnered great interest due to their expressive power. In particular, Kwon and Agha [7] introduced LTLC, an extension of conventional linear temporal logic for specifying properties of discrete-time linear systems, and described LTLC model checking that allows a sequence of control inputs to be automatically computed such that a complex control objective expressed in LTLC is satisfied. To make LTLC model checking decidable, its scope is limited to systems that reach a steady state in finite time. Karaman et al. [8] proposed a method based on mixed integer linear programming to incorporate temporal logic in control. The interaction with environments, however, was not taken into consideration.

The development of language equivalence and bisimulation notions allows abstraction of the continuous component of the system to a purely discrete model while preserving all the desired properties [9]. This subsequently provides a hierarchical approach to system design. In particular, a two-layer design is common and widely used in the area of planning and control [10], [11], [12], [13], [14]. In the first layer, a discrete planner plans, in the abstracted discrete domain, a set of transitions of the system to ensure the satisfaction of the desired properties. This abstract plan is then continuously implemented by a continuous controller in the second layer. Simulations/bisimulations provide the proof that the continuous execution preserves the desired properties. One of the main challenges of this approach is in the abstraction of continuous, infinite-state systems into equivalent (in the simulation sense) finite state models. Special cases of fully actuated ($\dot{x} = u$), kinematic ($\dot{x} = A(x)u$), piecewise affine (PWA) and discrete-time controllable linear systems have been studied in [10], [11], [12] and [13] respectively. Reference [14] deals with more general dynamics by using the notions of approximate simulation and simulation functions [15]. However, similar to a Lyapunov function, a simulation function can be difficult to compute.

Another main challenge of the two-layer approach is the computational complexity in the synthesis of discrete planners. Although it has been shown that for a certain class of properties, known as *Generalized Reactivity(1)*, a discrete planner can be automatically computed in polynomial time [1], the applications of the synthesis tool are limited to small problems due to the state explosion issue.

This paper partially addresses both of the aforementioned challenges. First, we present a fully automated approach to construct a finite state abstraction of a system with PWA dynamics. A notion of reachability is defined that is sufficient to ensure that the continuous execution preserves the correctness of the discrete plan. The requirement of a bisimulation abstraction is then relaxed to a simulation abstraction that is enforced by restricting the set of discrete plans to those satisfying the reachability relation, which can be established by solving a multi-parametric programming problem. The Multi-Parametric Toolbox [4] provides an off-the-shelf computational machinery that enables the multi-parametric programming problem to be solved in an automated fashion. As opposed to the approach proposed by

This work is partially supported by AFOSR and the Boeing Corporation.

The authors are with the Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA. {nok, utopcu, murray}@cds.caltech.edu

Kloetzer and Belta [12] where the state space is partitioned based solely on the linear predicates appearing in the desired properties, our approach allows refinement of the predicate-based partition of the state space. This subsequently reduces the conservativeness of simulation abstractions and enables us to identify the set of initial states starting from which a control law that ensures the satisfaction of the desired properties cannot be found and allows the synthesis problem to be solved assuming that the system starts from other states. Together with the digital design synthesis tool [1], our technique allows automatic design of dynamical systems that satisfy a wide range of properties expressed in temporal logic, taking into account the interaction with potentially adversarial environments.

Second, to partially address the state explosion problem in digital design synthesis, we introduce a receding horizon scheme for executing finite state automata while ensuring system correctness. This allows the synthesis to be performed on a smaller domain and thus potentially substantially reduce the size of the synthesis problem. Although the proposed approach is not completely automatic, to the authors' knowledge, it is the first time that a receding horizon technique is applied to finite state automata and ensures more sophisticated properties than stability, safety and convergence.

II. PRELIMINARIES

We use linear temporal logic (LTL) to describe the desired properties of the system. Given an LTL formula, we want to construct a finite state automaton, which can be thought of as a graph with a finite number of nodes (representing the states of the system) and edges (representing the transitions between states), such that the state transitions in the automaton ensure the correctness of the system. In this section, we briefly describe the definition of LTL and the synthesis of a finite state automaton which satisfies a given LTL formula.

A. Terminology and Notations

Definition 1: A system consists of a set V of variables. The *domain* of V, denoted by $dom(V)$, is the set of valuations of V. A *state* of the system is an element $v \in dom(V)$.

Definition 2: A *finite transition system* is a tuple $\mathbb{D} = (\mathcal{V}, \rightarrow)$ where $\mathcal{V}$ is a finite set of states, and $\rightarrow \subseteq \mathcal{V} \times \mathcal{V}$ is a transition relation. Given states $\nu_i, \nu_j \in \mathcal{V}$, we write $\nu_i \rightarrow \nu_j$ if there is a transition from ν_i to ν_j.

Definition 3: An *atomic proposition* is a statement on system variables v that has a unique truth value ($True$ or $False$) for a given value of v. Let $v \in dom(V)$ be a state of the system and π be an atomic proposition. We write $v \Vdash \pi$ if π is $True$ at the state v. Otherwise, we write $v \nVdash \pi$.

Definition 4: An *execution* of a discrete-time system is an infinite sequence of the states of the system over a particular run, i.e., an execution σ can be written as $\sigma = v_0 v_1 v_2 \ldots$ where for each $t \geq 0$, $v_t \in dom(V)$ is the state of the system at time t.

B. Linear Temporal Logic

The use of linear temporal logic (LTL) as a specification language was introduced by Pnueli [16], [17]. LTL is built up from a set of atomic propositions, the logic connectives ($\neg$, $\vee$, $\wedge$, $\Longrightarrow$), and the temporal modal operators ($\bigcirc$, $\square$, $\diamond$, $\mathcal{U}$ which are read as "next," "always," "eventually," and "until," respectively). An LTL formula is defined inductively as follows: (1) any atomic proposition π is an LTL formula; and (2) given an LTL formula φ and ψ, the following are also LTL formulas: $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$ and $\varphi \,\mathcal{U}\, \psi$. Other operators can be defined as follows: $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$, $\varphi \Longrightarrow \psi = \neg\varphi \vee \psi$, $\diamond\varphi = True \,\mathcal{U}\, \varphi$, and $\square\varphi = \neg\diamond\neg\varphi$. A *propositional* formula is one that does not include temporal operators. Given a set of LTL formulas $\varphi_1, \ldots, \varphi_n$, their *boolean combination* is an LTL formula formed by joining $\varphi_1, \ldots, \varphi_n$ with logic connectives.

Semantics of LTL: An LTL formula is interpreted over an infinite sequence of states. Given an execution $\sigma = v_0 v_1 v_2 \ldots$ and an LTL formula φ, we say that φ *holds at position* $i \geq 0$ of σ, written $v_i \models \sigma$, if and only if φ holds for the remainder of the execution σ starting at position i. The semantics of LTL is defined inductively as follows: 1) For an atomic proposition π, $v_i \models \pi$ iff $v_i \Vdash \pi$; 2) $v_i \models \neg\varphi$ iff $v_i \not\models \varphi$; 3) $v_i \models \varphi \vee \psi$ iff $v_i \models \varphi$ or $v_i \models \psi$; 4) $v_i \models \bigcirc\varphi$ iff $v_{i+1} \models \varphi$; and 5) $v_i \models \varphi \,\mathcal{U}\, \psi$ iff $\exists j \geq i, v_j \models \psi$ and $\forall k \in [i, j), v_k \models \varphi$. Based on this definition, $\square\varphi$ holds at position i iff φ holds at every position in σ starting at position i, and $\diamond\varphi$ holds at position i iff φ holds at some position $j \geq i$ in σ.

Definition 5: An execution $\sigma = v_0 v_1 v_2 \ldots$ *satisfies* φ, denoted by $\sigma \models \varphi$, if $v_0 \models \varphi$.

Definition 6: Let Σ be the set of all executions of a system. The system is said to be *correct* with respect to its specification φ, written $\Sigma \models \varphi$, if all its executions satisfy φ, that is, $(\Sigma \models \varphi) \iff \big(\forall\sigma, (\sigma \in \Sigma) \implies (\sigma \models \varphi)\big)$.

C. Synthesis of Finite State Automata

In many applications, systems need to interact with their environments and whether they satisfy the desired properties depends on what the environments do. For example, whether an autonomous car exhibits the correct behavior at an intersection depends on the behavior of other cars at the intersection, e.g. which car gets to the intersection first, etc. In this section, we informally describe the work of Piterman, et al. [1]. We refer the reader to [1] and references therein for the detailed discussion of automatic synthesis of a finite state automaton from its specification.

From Definition 6, for a system to be correct, its specification φ must be satisfied regardless of what the environment does. Thus, the environment can be treated as adversary and the synthesis problem can be viewed as a two-player game between the system and the environment: the environment attempts to falsify φ while the system attempts to satisfy φ. We say that φ is *realizable* if the system can satisfy φ no matter what the environment does.

For a specification of the form

$$\big(\bigwedge_{i \in I} \square\diamond\varphi_i\big) \implies \big(\bigwedge_{j \in J} \square\diamond\psi_j\big), \tag{1}$$

known as *Generalized Reactivity(1)*, Piterman et al. shows that checking its realizability and synthesizing the corresponding automaton can be performed in polynomial time.

In particular, we are interested in a specification of the form

$$\varphi = (\varphi_e \Longrightarrow \varphi_s) \tag{2}$$

where roughly speaking, φ_e characterizes the initial states of the system and the assumptions on the environment and φ_s describes the correct behavior of the system, including the valid transitions the system can make. We refer the reader to [1] for precise definitions of φ_e and φ_s. Note that since $\varphi_e \Longrightarrow \varphi_s$ is satisfied whenever φ_e is *False*, if the assumptions of the environment or the initial state of the system violate φ_e, the correct behavior φ_s of the system is not ensured, even though the specification φ is satisfied.

If the specification is realizable, the synthesis tool generates a finite state automaton that represents a set of transitions the system should follow in order to satisfy φ. Otherwise, it provides an initial state of the system starting from which there exists a set of moves of the environment such that the system cannot satisfy φ. The knowledge of the realizability of the specification is useful since it provides information about the conditions under which the system will fail to satisfy its desired properties.

The main limitation of the synthesis of finite state automata is the state explosion problem. In the worst case, the resulting automaton may contain all the possible states of the system. For example, if the system has 10 variables, each can take any value in $\{1, \ldots, 10\}$, then there may be as many as 10^{10} nodes in the automaton.

III. PROBLEM FORMULATION

Consider a system $\mathbb{S}$ with a set of variables $V = S \cup E$ where S and E represent, respectively, the set of variables controlled by the system and the set of variables controlled by the environment. For example, for an obstacle avoidance problem where a robot needs to navigate an environment populated with obstacles, S may include the state of the robot while E may include the positions of obstacles. The domain of V is therefore given by $dom(V) = dom(S) \times dom(E)$ and a state of the system can be written as $v = (s, e)$ where $s \in dom(S)$ and $e \in dom(E)$. Throughout the paper, we call s the *controlled* state and e the *environment* state.

Assume that the controlled state evolves according to the following discrete-time piecewise-affine (PWA) dynamics[1]:

$$\begin{array}{rcl} s[t+1] & = & A_k s[t] + B_k u[t] + C_k \text{ if } (s[t], u[t]) \in \Omega_k \\ u[t] & \in & U \end{array} \tag{3}$$

where $k \in \{1, \ldots, N_{PWA}\}$, N_{PWA} is the number of regions in the PWA partition, U is the set of admissible control inputs, $\{\Omega_1, \ldots, \Omega_{N_{PWA}}\}$ is a polyhedral partition of $dom(S) \times U$, for any natural number t, $s[t] \in dom(S)$ is the controlled state at time t and u is the control signal.

Let Π be a finite set of atomic propositions of variables from V and φ be an LTL specification built from Π. Suppose φ is a *Generalized Reactivity(1)* formula of the form (2) and can be expressed without the next operator ($\bigcirc$).[2] We want to design a controller that ensures that any execution $\sigma = v_0 v_1 \ldots$ of the system satisfies φ where for each natural number t, $v_t \in dom(V)$ is the state of the system at time t.

IV. HIERARCHICAL APPROACH

In general, constructing a controller that ensures that any execution of the system satisfies the specification φ while respecting the dynamics (3) is hard since both the adversarial nature of the environment and the dynamics of the system need to be taken into account. To separate the concern of the environment from the concern of the dynamics, we apply a hierarchical approach to solve the problem defined in Section III. That is, we decompose the problem into (a) designing a discrete planner that computes a discrete plan satisfying the specification φ regardless of what the environment does and (b) designing a continuous controller that implements the discrete plan while ensuring that the evolution of the system satisfies the dynamics (3).

The discrete planner can be automatically synthesized using the digital design synthesis tool [1] as described in Section II-C. However, since the synthesis algorithm requires a finite domain, the system $\mathbb{S}$ must be abstracted to a finite transition system. To construct a finite transition system $\mathbb{D}$ from $\mathbb{S}$, we first partition $dom(S)$ and $dom(E)$, as in [10], [12], into a finite number of equivalence classes or cells $\mathcal{S}$ and $\mathcal{E}$, respectively, such that the partition is *proposition preserving* [9]. Roughly speaking, this means that for any atomic proposition $\pi \in \Pi$ and any states v_1 and v_2 that belong to the same cell in the partition, if v_1 satisfies π, then v_2 also satisfies π. We denote the resulting discrete domain of the system by $\mathcal{V} = \mathcal{S} \times \mathcal{E}$. Throughout the paper, we call $v \in dom(V)$ a *continuous state* and $\nu \in \mathcal{V}$ a *discrete state* of the system. For a discrete state $\nu \in \mathcal{V}$, we say that ν satisfies an atomic proposition $\pi \in \Pi$, denoted by $\nu \Vdash_d \pi$, if and only if there exists a continuous state v contained in the cell labeled by ν such that $v \Vdash \pi$. Given an infinite sequence of discrete states $\sigma_d = \nu_0 \nu_1 \nu_2 \ldots$ and an LTL formula φ built from Π, we say that φ holds at position $i \geq 0$ of σ_d, written $\nu_i \models_d \varphi$, if and only if φ holds for the remainder of σ_d starting at position i. With these definitions, the semantics of LTL for a sequence of discrete states can be derived from the general semantics of LTL defined in Section II-B.

Next, we need to determine the transition relations $\rightarrow$ of $\mathbb{D}$. Since constructing a bisimulation partition for a general system with PWA dynamics is hard and such a partition may not be finite, in this section, we relax the requirement that the partition is bisimulation and define the notion of *reachability* that is sufficient (but not necessary) to guarantee that if a discrete controlled state $\mathcal{S}_j$ is reachable from $\mathcal{S}_i$, the transition from $\mathcal{S}_i$ to $\mathcal{S}_j$ can be continuously *implemented* or *simulated* by a continuous controller. (See, for example, [19] for the exact definition.) A computational scheme that provides a sufficient condition for reachability between two discrete controlled states and subsequently refines the state space partition is also presented in Section IV-B.

A. Reachability

Let $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ be a set of discrete controlled states. We define a map $T_s : dom(S) \rightarrow \mathcal{S}$ that sends a continuous controlled state to a discrete controlled state of

[1] We restrict ourselves to PWA dynamics for computational reasons. Our framework straightforwardly generalizes to nonlinear dynamics.

[2] This assumption is sufficient to ensure that φ is stutter invariant. See, for example, [18] for more detail.

its equivalence class. That is, $T_s^{-1}(\mathcal{S}_i) \subseteq dom(S)$ is a set of all the continuous controlled states contained in the cell labeled by $\mathcal{S}_i$ and $\{T_s^{-1}(\mathcal{S}_i), \ldots, T_s^{-1}(\mathcal{S}_n)\}$ is the partition of $dom(S)$. We define the reachability relation, denoted by $\rightsquigarrow$, as follows: a discrete state $\mathcal{S}_j$ is reachable from a discrete state $\mathcal{S}_i$, written $\mathcal{S}_i \rightsquigarrow \mathcal{S}_j$, only if starting from any point $s[0] \in T_s^{-1}(\mathcal{S}_i)$, there exists a control law $u \in U$ that takes the system (3) to a point $s[N] \in T_s^{-1}(\mathcal{S}_j)$ satisfying the constraint $s[t] \in T_s^{-1}(\mathcal{S}_i) \cup T_s^{-1}(\mathcal{S}_j), \forall t \in \{0, \ldots, N\}$ for some horizon length N. Note that this is stronger than the usual definition of reachability [20]. We write $\mathcal{S}_i \not\rightsquigarrow \mathcal{S}_j$ if $\mathcal{S}_j$ is not reachable from $\mathcal{S}_i$.

In general, for two discrete states $\mathcal{S}_i$ and $\mathcal{S}_j$, verifying the reachability relation $\mathcal{S}_i \rightsquigarrow \mathcal{S}_j$ is hard. Therefore, we resort to a heuristic based on the following optimal control problem: Given discrete controlled states $\mathcal{S}_i, \mathcal{S}_j \in \mathcal{S}$, the set of admissible control inputs U, the matrices A_k and B_k as in (3), a horizon length $N \geq 0$ and the cost matrices $P_N, Q \succeq 0$ and $R \succ 0$, solve

$$\begin{array}{ll}\min\limits_{u[0],\ldots,u[N-1]} & \|P_N \hat{s}[N]\|_2 + \sum\limits_{t=0}^{N-1} \|Q\hat{s}[t]\|_2 + \|Ru[t]\|_2 \\ \text{s.t.} & s[N] \in T_s^{-1}(\mathcal{S}_j), \quad s[0] \in dom(S) \\ & s[t+1] = A_k s[t] + B_k u[t] \text{ if } (s[t], u[t]) \in \Omega_k \\ & u[t] \in U \\ & s[t] \in T_s^{-1}(\mathcal{S}_i) \cup T_s^{-1}(\mathcal{S}_j) \\ & \forall t \in \{0, \ldots, N-1\}\end{array} \tag{4}$$

where for any $t \in \{0, \ldots, N\}$, $\hat{s}[t] = s[t] - s_j$ for some chosen $s_j \in T_s^{-1}(\mathcal{S}_j)$ (e.g. s_j may be the center of the cell labeled by $\mathcal{S}_j$). Note that (4) is a finite horizon optimal control problem. Furthermore, one can consider the problem in (4) as a family of problems parametrized by $s[0]$ and it can be regarded as a multi-parametric programming problem [3]. For the case where $T_s^{-1}(\mathcal{S}_i)$, $T_s^{-1}(\mathcal{S}_j)$, and U are polytopic sets, i.e., sets defined by affine inequalities, the explicit solution for this multi-parametric programming problem (i.e., the sequence of control inputs $u[0], \ldots, u[N-1]$ as a function of $s[0]$ and the set $\mathcal{P}_{i,j} \subseteq T_s^{-1}(\mathcal{S}_i) \cup T_s^{-1}(\mathcal{S}_j)$ such that (4) is feasible for all $s[0] \in \mathcal{P}_{i,j}$) can be computed using the Multi-Parametric Toolbox [4]. We refer the reader to [21] for a detailed discussion on how this multi-parametric programming problem can be solved. An example of a set $\mathcal{P}_{i,j}$ along with $\mathcal{S}_i$ and $\mathcal{S}_j$ is shown in Figure 1.

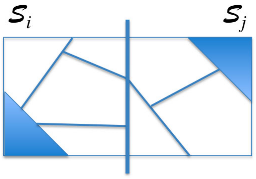

Fig. 1. An example of a set $\mathcal{P}_{i,j}$ represented by the unshaded region. For any $s[0]$ in the shaded region, the optimal control problem (4) is infeasible. Different unshaded regions have different associated controllers. For more detail, see [4].

B. State Space Discretization

In general, given the previous partition of $dom(S)$ and any $i, j \in \{1, \ldots, n\}$, the reachability relation between $\mathcal{S}_i$ and $\mathcal{S}_j$ may not be established through the solution of the multi-parametric programming problem (4) since $T_s^{-1}(\mathcal{S}_i)$ is not necessarily covered by $\mathcal{P}_{i,j}$ (due to the constraints on u and a specific choice of the finite horizon N). This section describes a state space discretization scheme based on the reachability relation defined earlier to increase the number of valid discrete state transitions of $\mathbb{D}$. The underlying idea is that for each $\mathcal{S}_i$ and $\mathcal{S}_j$, we determine $\mathcal{P}_{i,j}$ such that for any $s[0] \in \mathcal{P}_{i,j}$, the problem in (4) is feasible. Then, we partition $T_s^{-1}(\mathcal{S}_i)$ into $T_s^{-1}(\mathcal{S}_i) \cap \mathcal{P}_{i,j}$, labeled by $\mathcal{S}_{i,1}$ and $T_s^{-1}(\mathcal{S}_i) \setminus \mathcal{P}_{i,j}$, labeled by $T_s^{-1}(S_{i,2})$ and obtain the following reachability relations: $\mathcal{S}_{i,1} \rightsquigarrow \mathcal{S}_j$ and $\mathcal{S}_{i,2} \not\rightsquigarrow \mathcal{S}_j$.

TABLE I

Discretization Algorithm

```
Discretization Algorithm
input: The lower bound on cell volume ($Vol_{min}$), the parameters $A_k$,
    $B_k$, $\Omega_k$, $U$, $N$, $P_N$, $Q$, $R$ of the multi-parametric programming
    problem, and the original partition ($\{T_s^{-1}(\mathcal{S}_i) \mid i \in \{1, \ldots, n\}\}$)
output: The new partition $sol$

$sol = \{T_s^{-1}(\mathcal{S}_i) \,|\, i \in \{1, \ldots, n\}\}$; $IJ = \{(i,j) \,|\, i, j \in \{1, \ldots, n\}\}$;
while ($size(IJ) > 0$)
  Pick an $(i, j) \in IJ$;
  Solve the multi-parametric programming problem for $\mathcal{P}_{i,j}$;
  if ($volume(sol[i] \cap \mathcal{P}_{i,j}) > Vol_{min}$ and
      $volume(sol[i] \setminus \mathcal{P}_{i,j}) > Vol_{min}$) then
    Replace $sol[i]$ with $sol[i] \cap \mathcal{P}_{i,j}$ and add $sol[i] \setminus \mathcal{P}_{i,j}$ to $sol$;
    For each $k \in \{1, \ldots, size(sol)\}$, add $(i,k)$, $(k,i)$,
        $(size(sol), k)$ and $(k, size(sol))$ to $IJ$;
  else
    Remove $(i, j)$ from $IJ$;
  endif
endwhile
```

Discretization Algorithm: Pick a natural number N and the cost matrices P_N, Q and R. Define a lower bound Vol_{min} on the volume of each cell in the new partition. Starting with a pair (i, j) where $i, j \in \{1, \ldots, n\}$, determine the set $\mathcal{P}_{i,j}$ such that for any $s[0] \in \mathcal{P}_{i,j}$, the problem in (4) is feasible. If the volumes of both $T_s^{-1}(\mathcal{S}_i) \cap \mathcal{P}_{i,j}$ and $T_s^{-1}(\mathcal{S}_i) \setminus \mathcal{P}_{i,j}$ are greater than Vol_{min}, then partition $T_s^{-1}(\mathcal{S}_i)$ into $T_s^{-1}(\mathcal{S}_i) \cap \mathcal{P}_{i,j}$ and $T_s^{-1}(\mathcal{S}_i) \setminus \mathcal{P}_{i,j}$. Repeat this process until none of the cells can be partitioned. Table I shows the pseudo-code of the algorithm.

Remark 1: Vol_{min} only provides a terminating criterion for the proposed algorithm. Other criteria such as the maximum number of iterations can be used as well.

Remark 2: The proposed discretization algorithm terminates when no cell can be partitioned such that the volumes of the two resulting new cells are both greater than Vol_{min}. Larger Vol_{min} causes the algorithm to terminate sooner.

Remark 3: The point at which the algorithm terminates affects the reachability between discrete controlled states of the new partition and as a result, affects the realizability of the specification. Generally, a coarse partition makes the specification unrealizable but a fine partition causes state space explosion. A way to decide when to terminate the algorithm is to start with a coarse partition and keep refining it until the specification is realizable.

C. Correctness of the System

Let $\mathcal{S}' = \{\mathcal{S}'_1, \mathcal{S}'_2, \ldots, \mathcal{S}'_m\}$ be the set of all the discrete controlled states corresponding to the resulting partition of $dom(S)$ after applying the discretization algorithm proposed in Section IV-B. Since the partition obtained from the proposed algorithm is a subpartition of $\{T_s^{-1}(\mathcal{S}_1), \ldots, T_s^{-1}(\mathcal{S}_n)\}$ and $\mathcal{V} = \mathcal{S} \times \mathcal{E}$ is proposition preserving, it is trivial to show that $\mathcal{V}' = \mathcal{S}' \times \mathcal{E}$ is also proposition preserving. We define the finite transition system $\mathbb{D}$ that serves as the abstract model of $\mathbb{S}$ as follows: $\mathcal{V}' = \mathcal{S}' \times \mathcal{E}$ is the set of states of $\mathbb{D}$ and for any two states $\mathcal{V}_i = (\mathcal{S}'_{is}, \mathcal{E}_{ie})$ and $\mathcal{V}_j = (\mathcal{S}'_{js}, \mathcal{E}_{je})$, $\mathcal{V}_i \rightarrow \mathcal{V}_j$ (i.e. there exists a transition from $\mathcal{V}_i$ to $\mathcal{V}_j$) only if $\mathcal{S}'_{is} \rightsquigarrow \mathcal{S}'_{js}$. Using the abstract model $\mathbb{D}$, a discrete planner that guarantees the satisfaction of φ while ensuring that the discrete plans are restricted to those satisfying the reachability relations can be automatically synthesized using the digital design synthesis tool as described in Section II-C.

From the stutter invariant property of φ, the formulation of the optimal control problem (4) and the proposition preserving property of $\mathcal{V}'$, it is straightforward to prove the following proposition.

Proposition 1: Let $\sigma_d = \nu_0 \nu_1 \ldots$ be an infinite sequence of discrete states of $\mathbb{D}$ where for each natural number k, $\nu_k \rightarrow \nu_{k+1}$, $\nu_k = (\rho_k, \epsilon_k)$, $\rho_k \in \mathcal{S}'$ is the discrete controlled state and $\epsilon_k \in \mathcal{E}$ is the discrete environment state. If $\sigma_d \models_d \varphi$, then by applying a sequence of control laws, each corresponding to the solution of (4) with $\mathcal{S}_i = \rho_k$ and $\mathcal{S}_j = \rho_{k+1}$, the infinite sequence of continuous states $\sigma = v_0 v_1 v_2 \ldots$ satisfies φ.

V. RECEDING HORIZON STRATEGY

As discussed in Section II-C, automatic synthesis of finite state automata from their LTL specifications [1] suffers from the state explosion problem. In many applications, however, it is not necessary to plan for the whole execution, taking into account all the possible behaviors of the environment since a state that is very far from the current state of the system typically does not affect the near future plan. For example, consider a robot motion planning problem where the robot has to travel 100 kilometers. Under certain conditions, it may be sufficient to only plan out an execution for 500 meters and implement it in a receding horizon fashion, i.e., re-compute the plan as the robot moves. In this section, we present a sufficient condition and a receding horizon scheme that allows the synthesis to be performed on a smaller domain; thus, substantially reduces the number of states (or nodes) of the automaton while still ensuring the system correctness.

We consider a subclass of *Generalized Reactivity(1)* formulas (1): (a) let ψ_{init} be a propositional formula of variables from V which characterizes the initial state of the system; (b) let ψ_e be a boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_e^t$ where ψ_e^t is a propositional formula of variables from E which describes the assumptions on the transitions of environment states; (c) let ψ_s be a boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_s^t$ where ψ_s^t is a propositional formula of variables from V which describes the constraints on the transitions of discrete controlled states; and (d) let ψ_g be a propositional formula of variables from V. We assume that the corresponding (stronger) specification for $\mathbb{D}$ is given by[3]

$$\varphi_{\mathbb{D}} = (\psi_{init} \wedge \Box\psi_e) \implies (\Box\psi_s \wedge \Diamond\psi_g) \qquad (5)$$

where $\Box\psi_s$ and $\Diamond\psi_g$ express the safety and the progress properties of the system. Let $\mathcal{V}' = \mathcal{S}' \times \mathcal{E}$ be the discrete domain of the system after applying the discretization algorithm presented in Section IV. Similar to the map T_s for the controlled states defined in Section IV, we let $T : dom(V) \rightarrow \mathcal{V}'$ be a map that sends a continuous state to a discrete state of its equivalence class, i.e. for each $\mathcal{V}_i \in \mathcal{V}'$, $T^{-1}(\mathcal{V}_i) \in dom(V)$ is the set of all the continuous states contained in the cell labeled by $\mathcal{V}_i$ and $\{T^{-1}(\mathcal{V}_i) \mid \mathcal{V}_i \in \mathcal{V}'\}$ is a partition of $dom(V)$.

Suppose there exists a collection of disjoint subsets $\mathcal{W}_0, \ldots, \mathcal{W}_p$ of $\mathcal{V}'$ such that (a) $\mathcal{W}_0 \cup \mathcal{W}_1 \cup \ldots \cup \mathcal{W}_p = \mathcal{V}'$, (b) ψ_g is satisfied for any $v \in \bigcup_{\mathcal{V}_i \in \mathcal{W}_0} T^{-1}(\mathcal{V}_i)$, i.e., $\mathcal{W}_0$ is the set of the final states, and (c) $(\{\mathcal{W}_0, \ldots, \mathcal{W}_p\}, \preceq_{\psi_g})$ is a partially ordered set. By an abuse of notation, for each $i \in \{0, \ldots, p\}$, we let $T^{-1}(\mathcal{W}_i) = \bigcup_{\mathcal{V}_k \in \mathcal{W}_i} T^{-1}(\mathcal{V}_k)$, i.e., $T^{-1}(\mathcal{W}_i)$ is the set of all the continuous states contained in the cells that belong to the set $\mathcal{W}_i$. Further assume that there exists a propositional formula Φ of variables from V and for each $i \in \{0, \ldots, p\}$, there exist $g_i \in \{0, \ldots, p\}$ and a subset $\mathcal{D}_i$ of $dom(S)$ satisfying the following conditions:

(1) $\psi_{init} \implies \Phi$ is a tautology, i.e., any state that satisfies ψ_{init} also satisfies Φ,
(2) $T^{-1}(\mathcal{W}_i)$, $T^{-1}(\mathcal{W}_{g_i}) \subseteq \mathcal{D}_i \times dom(E)$, and
(3) $\mathcal{W}_{g_i} \preceq_{\psi_g} \mathcal{W}_i$ and for each $i \neq 0$, $\mathcal{W}_{g_i} \prec_{\psi_g} \mathcal{W}_i$

such that

$$\begin{aligned}\Psi_i = \big((v \in T^{-1}(\mathcal{W}_i)) \wedge \Phi \wedge \Box\psi_e\big) \\ \implies \big(\Box\psi_s \wedge \Diamond(v \in T^{-1}(\mathcal{W}_{g_i})) \wedge \Box\Phi\big)\end{aligned} \qquad (6)$$

is realizable with the domain of S restricted to $\mathcal{D}_i$.

For $i \in \{0, \ldots, p\}$, let $\mathcal{A}_i$ be an automaton that satisfies Ψ_i. Since in the synthesis of $\mathcal{A}_i$, the domain of S is restricted to $\mathcal{D}_i$, this can substantially reduce the number of states in the automaton, especially when the size of $\mathcal{D}_i$ is much smaller than the size of $dom(S)$.

Receding Horizon Strategy: Starting from the state v_0, pick an automaton $\mathcal{A}_i$ such that $v_0 \in T^{-1}(\mathcal{W}_i)$ and execute $\mathcal{A}_i$ until the system reaches the state $v \in T^{-1}(\mathcal{W}_j)$ where $\mathcal{W}_j \prec_{\psi_g} \mathcal{W}_i$, at which point, switch to the automaton $\mathcal{A}_j$. Repeat this process until $\mathcal{A}_0$ is executed.

Theorem 1: Suppose for each $i \in \{0, \ldots, p\}$, Ψ_i is realizable. Then the proposed receding horizon strategy ensures the correctness of the system.

Proof: Consider an arbitrary execution σ of the system that satisfies the formula to the left of $\implies$ in (5). From the tautology of $\psi_{init} \implies \Phi$, it is easy to show that if σ starts from $v \in T^{-1}(\mathcal{W}_i)$, then σ satisfies the formula to the left of $\implies$ in (6). Let $v_0 \in dom(V)$ be the initial state of the system. First, suppose $v_0 \in T^{-1}(\mathcal{W}_0)$. Then, the system always executes $\mathcal{A}_0$; thus, Ψ_0 ensures that σ satisfies (5). Next, suppose $v_0 \in T^{-1}(\mathcal{W}_i)$ where $i \neq 0$. Then, the system

[3] The specification of $\mathbb{D}$ is obtained by adding LTL formulas of the form $\psi_1 \implies \bigcirc\psi_2$ to the original specification φ which essentially restricts the valid state transitions to those satisfying the reachability relations as described in Section IV-C. Note that $\varphi_{\mathbb{D}}$ is generally not stutter invariant.

executes $\mathcal{A}_i$ and Ψ_i ensures that the safety property ψ_s holds at every position of σ up to and including position p_i at which the system switches the automaton and Φ holds at position p_i. In addition, since Ψ_i satisfies the progress property $\diamond(v \in T^{-1}(\mathcal{W}_{g_i}))$ where $\mathcal{W}_{g_i} \prec_{\psi_g} \mathcal{W}_i$, Ψ_i ensures that eventually the system reaches the state $v_j \in T^{-1}(\mathcal{W}_j)$ where $\mathcal{W}_j \prec_{\psi_g} \mathcal{W}_i$. According to the receding horizon scheme, the system switches the automaton at this state, i.e., v_j is the state of the system at position p_i of σ. Since $v_j \in T^{-1}(\mathcal{W}_j)$ and v_j satisfies Φ, σ satisfies the formula to the left of $\implies$ in (6). Using the previous argument, we get that Ψ_j ensures that the safety property ψ_s holds at every position of σ starting from position p_i up to and including position p_j at which the system switches the automaton and Φ holds at position p_j. By repeating this proof, we get that ψ_s holds at every position of σ and due to the finiteness of the set $\{\mathcal{W}_0, \ldots, \mathcal{W}_p\}$ and its partial order, eventually the automaton $\mathcal{A}_0$ is executed which ensures that σ satisfies the progress property $\diamond\psi_g$. ■

Remark 4: The propositional formula Φ (which can be viewed as an invariant of the system) adds a constraint on the initial state of the system assumed by each of the automata so that Ψ_i is realizable. One way to determine Φ is to start with $\Phi = True$ and check the realizability of the resulting Ψ_i. If for any $i \in \{0, \ldots, p\}$, Ψ_i is realizable, we are done. Otherwise, the synthesis process provides the initial state of the system starting from which there exists a set of moves of the environment such that the system cannot satisfy Ψ_i. This information provides guidelines for constructing Φ.

Remark 5: The partial order $(\{\mathcal{W}_0, \ldots, \mathcal{W}_p\}, \preceq_{\psi_g})$ essentially provides a measure of "closeness" to the goal (i.e. the set of the final states). Since each specification Ψ_i in (6) asserts that the system eventually reaches a state that is smaller in the partial order, it essentially ensures that each automaton $\mathcal{A}_i$ brings the system "closer" to the goal.

VI. EXAMPLE

We consider a point-mass omnidirectional vehicle navigating a straight road while avoiding obstacles and obeying certain traffic laws. It was shown in [22] that the nondimensional equations of motion of the vehicle are given by

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} \dot{x} \\ \dot{y} \\ \frac{2mL^2}{J}\dot{\theta} \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \\ q_\theta \end{bmatrix}, \qquad (7)$$

with the following constraints on the control efforts:

$$\forall t, q_x^2(t) + q_y^2(t) \leq \left(\frac{3 - |q_\theta(t)|}{2}\right)^2 \text{ and } |q_\theta(t)| \leq 3. \qquad (8)$$

Conservatively, we can set $|q_x(t)| \leq \sqrt{0.5}$, $|q_y(t)| \leq \sqrt{0.5}$ and $|q_\theta(t)| \leq 1$ so that the constraints (8) are decoupled.

In this section, we are only interested in the translational (x and y) components of the vehicle state. Discretizing the dynamics (7) with time step 0.1, we obtain the following discrete-time linear time-invariant state space model

$$\begin{bmatrix} z[t+1] \\ v_z[t+1] \end{bmatrix} = \begin{bmatrix} 1 & 0.0952 \\ 0 & 0.9048 \end{bmatrix} \begin{bmatrix} z[t] \\ v_z[t] \end{bmatrix} + \begin{bmatrix} 0.0048 \\ 0.0952 \end{bmatrix} q_z \qquad (9)$$

where z represents either x or y and v_z represents the rate of change in z. Let C_z be the domain of the vehicle state projected onto the (z, v_z) coordinates. We restrict the domain C_z to $[zmin, zmax] \times [-1, 1]$ and partition C_z as $C_z = \bigcup_{i \in \{zmin+1, \ldots, zmax\}} C_{z,i}$ where $C_{z,i} = [i-1, i] \times [-1, 1]$ as shown in Figure 2. Throughout the section, we call this partition the *original* partition of the domain C_z.

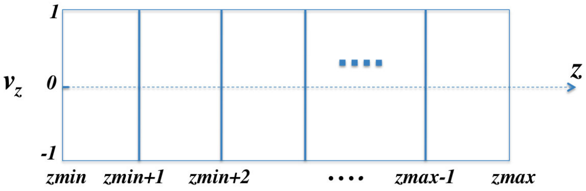

Fig. 2. The original partition of the domain C_z

We consider a road with 2 lanes, each of width 1, so we set $ymin = 0$ and $ymax = 2$. Since the vehicle dynamics are translationally invariant, without loss of generality we set $xmin = 0$ and $xmax = L$ where L is the length of the road.

For each $i \in \{1, \ldots, L\}$ and $j \in \{1, 2\}$, we define a Boolean variable $O_{i,j}$ that is assigned the value $True$ if and only if an obstacle is detected at some position $(x_o, y_o) \in [i-1, i] \times [j-1, j]$. The state of the system is therefore a tuple $(x, v_x, y, v_y, O_{1,1}, O_{1,2}, \ldots, O_{L,1}, O_{L,2})$ where $(x, v_x, y, v_y) \in [0, L] \times [-1, 1] \times [0, 2] \times [-1, 1]$ is the vehicle state or the controlled state and $(O_{1,1}, O_{1,2}, \ldots, O_{L,2}) \in \{0, 1\}^{2L}$ is the environment state.

A. System Specification

We assume that at the initial configuration, the vehicle is at least d_{obs} away from any obstacle and that the vehicle starts in the right lane. That is, ψ_{init} in (5) is defined as: for any $i \in \{1, \ldots, L\}$,

$$\left(x \in \bigcup_{k=i-d_{obs}}^{i+d_{obs}} C_{x,k} \implies (\neg O_{i,1} \wedge \neg O_{i,2})\right) \wedge y \in C_{y,1} \qquad (10)$$

The following properties are assumed for the environment.

1) An obstacle is detected before the vehicle gets too close to it. That is, there is a lower bound $d_{popup} \geq 0$ on the distance from the vehicle for which obstacle is allowed to instantly pop up. An LTL formula corresponding to this assumption is a conjunction of the following formula: for all $i \in \{1, \ldots, L\}$ and $k \in \{1, 2\}$,

$$\square\left(\left(x \in \bigcup_{j=i-d_{popup}}^{i+d_{popup}} C_{x,j} \wedge \neg O_{i,k}\right) \implies \square(\neg O_{i,k})\right) \qquad (11)$$

2) Sensing range is limited. That is, the vehicle cannot detect an obstacle that is away from it farther than $d_{sr} > d_{popup} \geq 0$. An LTL formula corresponding to this assumption is a conjunction of the following formula: for all $i \in \{1, \ldots, L\}$,

$$\square\left(x \in C_{x,i} \implies \bigwedge_{j > i+d_{sr}} (\neg O_{j,1} \wedge \neg O_{j,2})\right) \qquad (12)$$

3) The road is not blocked. That is, for any $i \in \{1, \ldots, L\}$,

$$\square\,(\neg O_{i,1} \vee \neg O_{i,2}) \qquad (13)$$

4) To make sure that the stay-in-lane requirement (see below) is achievable, we assume that an obstacle on the right lane does not disappear while the vehicle is in its vicinity. That is, for any $i \in \{1, \ldots, L\}$,

$$\square\left(\left(x \in \bigcup_{j=i-1}^{i+1} C_{x,j} \wedge O_{i,1}\right) \implies \square(O_{i,1})\right) \tag{14}$$

These assumptions can be relaxed so that they have the form (5) by replacing the inner $\square$ in (11) and (14) with $\bigcirc$.

Next, we define the desired safety property, $\square\psi_s$, as the conjunction of the following properties:

1) No collision, i.e., for any $i \in \{1, \ldots, L\}$ and $j \in \{1, 2\}$,

$$\square(O_{i,j} \implies \neg(x \in C_{x,i} \wedge y \in C_{y,j})) \tag{15}$$

2) The vehicle stays in the right lane unless there is an obstacle blocking the lane. That is, for any $i \in \{1, \ldots, L\}$,

$$\square((\neg O_{i,1} \wedge x \in C_{x,i}) \implies (y \in C_{y,1})) \tag{16}$$

Finally, we define $\psi_g = (x \in C_{x,L})$, i.e., we want to ensure that eventually the vehicle gets to the end of the road.

B. State Space Discretization

Since the dynamics and the constraints on the control efforts for the x and y components of the vehicle state are decoupled, we apply the discretization algorithm presented in Section IV for the x and y components separately for the sake of computational efficiency.[4] Since the vehicle dynamics (7) are translationally invariant, we can use similar partitions for all $C_{z,i}$. The discretization algorithm with horizon length $N = 10$ and $Vol_{min} = 0.1$ yields a partition with 11 cells $\{C^1_{z,i}, C^2_{z,i}, \ldots, C^{11}_{z,i}\}$ for each $C_{z,i}$ as shown in Fig. 3. For each $i \in \{zmin+1, \ldots, zmax\}$ and $j \in \{1, \ldots, 11\}$, we let $\mathcal{C}^j_{z,i}$ be the state label of cell $C^j_{z,i}$ and let $\mathcal{C}_{z,i} = \{\mathcal{C}^1_{z,i}, \ldots, \mathcal{C}^{11}_{z,i}\}$. A discrete state is therefore a tuple $(\nu_x, \nu_y, O_{1,1}, \ldots, O_{L,2})$ where $(\nu_x, \nu_y) \in \mathcal{C}_{x,i} \times \mathcal{C}_{y,i}$ is the discrete controlled state. Using MPT [4], the reachability between discrete controlled states can be determined and a controller associated with each reachable pair of them can be generated such that the resulting continuous execution implements the discrete transition between them. The specification of the resulting finite transition system can then be derived as discussed in Section IV-C.

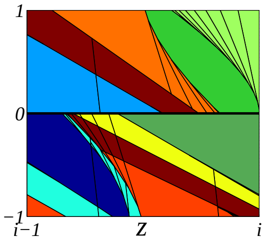

Fig. 3. The partition of each cell $C_{z,i}$ in the original partition of the domain C_z

C. Receding Horizon Formulation

Based on the new partition of the vehicle state space, there are the total of $242 \times L$ discrete vehicle states and $2^{2\times L}$ discrete environment states. Thus, in the worst case, the resulting automaton may have as many as $242 \times L \times 2^{2\times L}$ nodes. To avoid state explosion, we apply the receding horizon strategy proposed in Section V. The partial order structure is defined as $\mathcal{W}_i = \{(\nu_x, \nu_y, O_{1,1}, \ldots, O_{L,2}) \mid \nu_x \in \mathcal{C}_{x,L-i}\}$ and $\mathcal{W}_i \prec_{\psi_g} \mathcal{W}_j$ for any $i < j$.

Next, we follow the scheme in Remark 4 to find an invariant Φ. Starting with $\Phi = True$, we iteratively add, until Ψ_i as defined in (6) is realizable, a propositional formula to exclude the initial states starting from which there exists a set of moves of the environment such that the system cannot satisfy Ψ_i. A close examination of the resulting Φ reveals that Φ is essentially the conjunction of the following logics:

1) To ensure the progress property $\diamond\psi_g$, we need to assume that $\nu_x \notin \mathcal{X}_{notrans}$ and $\nu_y \notin \mathcal{Y}_{notrans}$ where $\mathcal{Z}_{notrans}$ is defined as: for any $\nu_z \in \mathcal{Z}_{notrans}$, $i \in \{zmin+1, \ldots, zmax\}$ and $j \in \{1, \ldots, 11\}$, $\nu_z \not\to \mathcal{C}^j_{z,i}$ and $\mathcal{Z}$ represent either $\mathcal{X}$ or $\mathcal{Y}$.
2) To ensure no collision, the vehicle cannot collide with an obstacle at the initial state.
3) Suppose $\nu_x \in \mathcal{C}_{x,i}$. To ensure no collision, if ν_y can only transition to $\nu'_y \in \mathcal{C}_{y,1}$, then either $O_{i,1}$ or $O_{i+1,1}$ is $False$. Similarly, if ν_y can only transition to $\nu'_y \in \mathcal{C}_{y,2}$, then either $O_{i,2}$ or $O_{i+1,2}$ is $False$. Similar reasoning can be derived for the case where $\nu_x \in \mathcal{C}_{x,i}$ such that it can only transition to $\nu'_x \in \mathcal{C}_{x,i+1}$ and for the case where it can only transition to $\nu'_x \in \mathcal{C}_{x,i}$.
4) To ensure the stay-in-lane property, the vehicle cannot be in the left lane unless there is an obstacle blocking the right lane at the initial state. In addition, the vehicle is never in the state $(\nu_x, \nu_y) \in \mathcal{C}_{x,i} \times \mathcal{C}_{y,1}$ which can only transition to $(\nu'_x, \nu'_y) \in \mathcal{C}_{x,i} \times \mathcal{C}_{y,2}$.
5) Suppose $\nu_x \in \mathcal{C}_{x,i}$ and $O_{i+1,1}$ is $False$. To ensure that the vehicle does not go to the left lane when the right lane is not blocked, it is not the case that $\nu_y \in \mathcal{C}_{y,1}$ which can only transition to $\nu'_y \in \mathcal{C}_{y,2}$. In addition, it is not the case that ν_x can only transition to $\nu'_x \in \mathcal{C}_{x,i+1}$ and $\nu_y \in \mathcal{C}_{y,2}$ which can only transition to $\nu'_y \in \mathcal{C}_{y,2}$.

With $d_{popup} = 1$ and the horizon length 2 (i.e. $g_i = i+2$), the specification (6) is realizable. In addition, if we let d_{obs} be greater than 1 and restrict the initial state of the system such that $\nu_x \notin \mathcal{X}_{notrans}$ and $\nu_y \notin \mathcal{Y}_{notrans}$, we get that $\psi_{init} \implies \Phi$ is a tautology.

D. Results

The synthesis was performed on a Pentium 4, 3.4 GHz computer with 4 Gb of memory. The computation time was 1230 seconds. The resulting automaton contains 2845 nodes. During the synthesis process, 96796 nodes were generated. Based on the authors experience, this particular computer crashes when approximately 97500 nodes are generated. Thus, this problem with horizon length 2 is as large as what the computer can handle. This means that without the receding horizon strategy, problems with the road of length greater than 3 cannot be solved.

[4] Before performing the discretization, we partition each $C_{z,i}$ into $\left(C^+_{z,i} \cup C^-_{z,i}\right)$ where $C^+_{z,i} = [i-1, i] \times [0, 1]$ and $C^-_{z,i} = [i-1, i] \times [-1, 0]$ to allow the possibility of enforcing other traffic laws such as disallowing reverse motion of the vehicle.

A simulation result with the road length of 30 is shown in Fig. 4. The polygons drawn in red are obstacles which are not known a priori. Notice that when there is no obstacle blocking the lane, the vehicle tries to stay as close to the lane boundary ($y = 1$) as possible. This is expected since to be able to avoid a pop up obstacle, due to the constraint on the admissible control inputs, the vehicle needs to stay close to the lane boundary to be able to change lane. To force the vehicle to stay close to the center of the lane, we need a finer partition of the road and extra LTL formula to ensure this property needs to be added to the system specification.

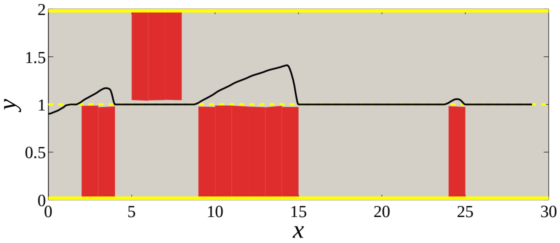

Fig. 4. Simulation result. The solid line is the trajectory of the vehicle. The polygons are obstacles discovered during the execution when the vehicle gets close enough to them.

VII. CONCLUSIONS AND FUTURE WORK

This paper illustrated how off-the-shelf tools from computer science and control can be integrated to allow automatic synthesis of complex dynamical systems that are guaranteed, by construction, to satisfy the desired properties expressed in temporal logic even in the presence of adversary (typically arising from changes in the environments). A receding horizon scheme for executing finite state automata was described that addresses the main limitation of the synthesis tool, the state explosion problem, assuming that the system has a certain partial order structure. The example showed that without the receding horizon scheme, the synthesis problem can be extremely computationally challenging.

Although the adversarial nature of the environment has been incorporated in the synthesis, the effects of disturbances and modeling errors have not yet been studied. To increase the robustness of the system, we plan to impose more conditions on the multi-parametric programming problem so that the continuous control law can be executed in a closed loop manner. In addition, the system specification needs to be modified to allow the possibility that the system may deviate from the plan due to disturbances and modeling errors.

Automatic or semi-automatic computation of an invariant Φ in the receding horizon scheme based on the information provided by the synthesis tool is also of interest. This direction sounds promising since, as described in the paper, Φ can be constructed by iteratively adding, until Ψ_i is realizable, a propositional formula to exclude the initial states of the system starting from which there exists a set of moves of the environment such that the system cannot satisfy Ψ_i.

VIII. ACKNOWLEDGMENTS

The authors gratefully acknowledge Hadas Kress-Gazit for the inspiring discussions on the automatic synthesis of finite state automata, for her suggestions and help with the synthesis tool, and for the code for generating an input to the synthesis tool and extracting its output; K. Mani Chandy and Knot Pipatsrisawat for the discussions and comments regarding the restriction of domain for the synthesis problem; and Vanessa Jönsson for thoughtful comments on the paper.

REFERENCES

[1] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Verification, Model Checking and Abstract Interpretation*, vol. 3855 of *Lecture Notes in Computer Science*, pp. 364 – 380, Springer-Verlag, 2006. Software available at http://www.wisdom.weizmann.ac.il/∼saar/synthesis/.

[2] R. M. Murray, J. Hauser, A. Jadbabaie, M. B. Milam, N. Petit, W. B. Dunbar, and R. Franz, "Online control customization via optimization-based control," in *Software-Enabled Control: Information Technology for Dynamical Systems*, pp. 149–174, Wiley-Interscience, 2003.

[3] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[4] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)," 2004. Available at http://control.ee.ethz.ch/∼mpt/.

[5] A. J. der Schaft and J. M. Schumacher, *Introduction to Hybrid Dynamical Systems*. London, UK: Springer-Verlag, 1999.

[6] J. Hespanha, "Stabilization through hybrid control," in *Encyclopedia of Life Support Systems (EOLSS)*, vol. Control Systems, Robotics, and Automation, 2004.

[7] Y. Kwon and G. Agha, "LTLC: Linear temporal logic for control," in *HSCC '08: Proceedings of the 11th international workshop on Hybrid Systems*, (Berlin, Heidelberg), pp. 316–329, Springer-Verlag, 2008.

[8] S. Karaman, R. G. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pp. 2117–2122, Dec. 2008.

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

[10] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Where's waldo? sensor-based temporal logic motion planning," *Robotics and Automation, 2007 IEEE International Conference on*, pp. 3116–3121, April 2007.

[11] D. Conner, H. Kress-Gazit, H. Choset, A. Rizzi, and G. Pappas, "Valet parking without a valet," *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 572–577, 2007.

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

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

[14] A. Girard and G. J. Pappas, "Brief paper: Hierarchical control system design using approximate simulation," *Automatica*, vol. 45, no. 2, pp. 566–571, 2009.

[15] A. Girard, A. A. Julius, and G. J. Pappas, "Approximate simulation relations for hybrid systems," *Discrete Event Dynamic Systems*, vol. 18, no. 2, pp. 163–179, 2008.

[16] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science*, pp. 46–57, IEEE, 1977.

[17] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems*. Springer-Verlag, 1992.

[18] D. Peled and T. Wilke, "Stutter-invariant temporal properties are expressible without the next-time operator," *Inf. Process. Lett.*, vol. 63, no. 5, pp. 243–246, 1997.

[19] H. Tanner and G. J. Pappas, "Simulation relations for discrete-time linear systems," in *Proceedings of the 15th IFAC World Congress on Automatic Control*, pp. 1302–1307, 2002.

[20] S. Prajna, *Optimization-based methods for nonlinear and hybrid systems verification*. PhD thesis, California Institute of Technology, 2005.

[21] F. Borrelli, *Constrained Optimal Control of Linear and Hybrid Systems*, vol. 290 of *Lecture Notes in Control and Information Sciences*. Springer, 2003.

[22] T. Kalmr-Nagy, R. D'Andrea, and P. Ganguly, "Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle," *Robotics and Autonomous Systems*, vol. 46, no. 1, pp. 47 – 64, 2004.