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 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 langauge of temporal logic. With its expressive power, a wider class of properties than safety and stability can be specified. The resulting system contains a discrete planner which plans, in the discrete domain, a set of transitions of the system to ensure the correct behaviors and a continuous controller which continuously implements the plan. For a system with lattice structure, we present an approach, based on a receding horizon scheme, to overcome computational difficulties in the synthesis of a discrete planner and allow more complex problems to be solved.**

I. INTRODUCTION

Recent advances in computer science, the development of a polynomial-time algorithm to construct finite state automata from their temporal logic specifications, allow automatic synthesis of digital designs so that a large class of properties including safety, guarantee and response are ensured even in the presence of adversary [1]. On the other hand, recent advances in control and abundance of computational resources allow continuous controllers to be designed in an automated fashion so that safety and stability properties are ensured 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. Furthermore, these systems may contain both continuous and discrete components. A grand challenge in this paradigm is to integrate the methods from the two communities such that automatic synthesis of such systems is possible.

Hybrid system theory has been developed to deal with systems with both discrete and continuous components. Control of hybrid systems has been studied by many researchers [5], [6]. However, properties of interest are limited to stability and safety. For the system to be able 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. A language of temporal logic has therefore garnered great interest due to its expressive power. MILP formulation has been used to incorporate temporal logic in control [7]. The interaction with potentially adversarial environments, however, is not taken into consideration.

This work is partially supported by AFOSR and Boeing Corporation.

T. Wongpiromsarn, U. Topcu and R.M. Murray are with the Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA nok@caltech.edu, utopcu@cds.caltech.edu, murray@cds.caltech.edu

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 [8]. This subsequently provides a hierarchical approach to system design. In the first layer, a discrete planner plans, in the 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. Bisimulations provide the (sufficient and necessary) proof that the continuous execution preserves the desired properties. This hierarchical approach has been applied in robot motion planning problems for a special case of fully actuated ($\dot{x} = u$) [9], [10] and kinematic ($\dot{x} = A(x)u$) [11] models. In [10], [11], the discrete planner is constructed using digital design synthesis tools [1]. The main limitation of such tools is the state explosion problem which restricts their applications to small problems.

The contribution of this paper is twofold. First, we extend the work of [10], [11] to a more complicated system with linear and piecewise affine (PWA) dynamics. The notion of reachability is defined which is sufficient to ensure that the continuous execution preserves the correctness of the discrete plan. The requirement of bisimulation abstraction is then relaxed by restricting the set of discrete plans to those satisfying the reachability relation which can be established by solving a multiparametric programming problem. The Multi-Parametric Toolbox [4] provides an off-the-shelf computational machinery which enables the multiparametric programming problem to be solved in an automated fashion. Together with the digital design synthesis synthesis tool [1], this allows automatic design of dynamical systems which satisfy a wide range of properties expressed in temporal logic, taken into account the adversary nature of the environments.

Second, to address the state explosion problem in the digital design synthesis, we present 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 space space of the synthesis problem. As a result, more complex problems can be handled.

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 resulting the state transitions in the automaton ensures 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 of a set of variables V. The *domain* of V, denoted by $dom(V)$, is the set of valuations of V. A *state* of the system is an element $v \in dom(V)$. For a state $v \in dom(V)$ and a set of states $\mathcal{U} \subseteq dom(V)$, their restrictions to a subset of variables $Z \subseteq V$ are denoted by $v \restriction Z$ and $\mathcal{U} \restriction Z$, respectively.

Definition 2: An *atomic proposition* is a statement on the system variables v which 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 is an atomic proposition. We write $v \Vdash \pi$ if π is $True$ at the state v. Otherwise, we write $v \nVdash \pi$.

Definition 3: A *transition system* is a tuple $(V, \Pi, \rightarrow, \vDash, V_0)$ where (a) V is a (possibly infinite) set of states, (b) Π is a finite set of atomic propositions, (c) $\rightarrow \subseteq V \times V$ is a transition relation, (d) $\vDash \ V \times \Pi$ is a satisfaction relation, and (e) $V_0 \subseteq V$ is a set of initial states.

Definition 4: An equivalence relation $\sim \subseteq V \times V$ on the domain V is *proposition preserving* if for all states $v_1, v_2 \in V$ and all propositions $\pi \in \Pi$, if $v_1 \sim q$ and $v_1 \Vdash \pi$, then $v_2 \Vdash \pi$.

Definition 5: For a transition system $\mathbb{T} = (V, \Pi, \rightarrow, \vDash, V_0)$, a proposition-preserving equivalence relation $\sim_B$ on V is a *bisimulation* of $\mathbb{T}$ if for all states $v_1, v_2 \in V$, if $v_1 \sim_B v_2$, then for all states $v_1' \in V$, if $v_1 \rightarrow v_1'$, there exists a state $v_2' \in V$ such that $v_2 \rightarrow v_2'$ and $v_2 \sim_B v_2'$.

Definition 6: An *execution* of the system is an infinite sequence of the states of the system over a particular run, i.e. an execution σ can be written as $\sigma = s_0 s_1 s_2 \ldots$ where $\forall i \geq 0, s_i$ is a state of the system.

B. Linear Temporal Logic

The use of linear temporal logic (LTL) as a specification language was introduced by Pnueli [12], [13]. LTL is built up from a set of atomic propositions, the logic connectives ($\neg$, $\vee$, $\wedge$, $\Longrightarrow$), and the temporal modal operators ($\bigcirc$, $\Box$, $\Diamond$, $\mathcal{U}$)[1]. An LTL formula is defined inductively as follows.

1) Any atomic proposition π is an LTL formula.
2) Given an LTL formula φ and ψ, the following are also LTL formulas: $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$ and $\varphi \,\mathcal{U}\, \psi$.

That is, in Backus Naur form:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi$$

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 $\Box\varphi = \neg\Diamond\neg\varphi$. 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 $\sigma = s_0 s_1 s_2 \ldots$ and an LTL formula φ, we say that φ *holds at position* $i \geq 0$ of σ, written $s_i \vDash \sigma$, if and only if φ holds for the remainder of the execution σ starting at position i. The semantics of the temporal logic formula is defined as follows:

1) For an atomic proposition π, $s_i \vDash \pi$ iff $s_i \Vdash p$
2) $s_i \vDash \neg\varphi$ iff $s_i \nvDash \varphi$
3) $s_i \vDash \psi \ \vee \ \varphi$ iff $s_i \vDash \psi$ or $s_i \vDash \varphi$
4) $s_i \vDash \bigcirc\varphi$ iff $s_{i+1} \vDash \varphi$
5) $s_i \vDash \psi \,\mathcal{U}\, \varphi$ iff $\exists j \geq i, s_j \vDash \varphi$ and $\forall k \in [i, j), s_k \vDash \psi$

Based on the above definition, $\Box\varphi$ 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 7: 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 $\Sigma \vDash \varphi$, if all its executions satisfy φ, that is,

$$(\Sigma \vDash \varphi) \iff (\forall\sigma, (\sigma \in \Sigma) \implies (\sigma \vDash \varphi)). \tag{1}$$

C. Synthesis of a Finite State Automaton

In many applications, systems need to interact with the environments and whether they satisfy the desired properties depends on what the environments do. In this section, we informally describe the work of Piterman, et al. on Synthesis of Reactive(1) Designs [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.

Based on Definition 7, for a system to be correct, its specification φ needs to be satisfied in all of its executions 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

$$(\bigwedge_{i \in I} \Box \Diamond \varphi_i) \implies (\bigwedge_{j \in J} \Box \Diamond \psi_j), \tag{2}$$

known as Generalized Reactivity(1) formulas, [1] 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

$$\varphi = (\varphi_e \Longrightarrow \varphi_s) \tag{3}$$

where roughly speaking, φ_e is an LTL formula which characterizes the initial states of the system and the assumptions of the environment and φ_s is an LTL formula which describes the correct behavior of the system, including the valid transitions the system can make. We refer the reader to [1] for a precise definition 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 initial state of the system violates φ_e, the system is not guaranteed to satisfy its correct behavior φ_s, even though the specification φ is satisfied.

If the specification is not realizable, the synthesis algorithm provides an initial state of the system for which there

[1] $\bigcirc$, $\Box$, $\Diamond$ and $\mathcal{U}$ are read as "next," "always," "eventually," and "until," respectively. Some authors also use release ($\mathcal{R}$) which can be defined as $\psi \,\mathcal{R}\, \varphi = \neg(\neg\psi \,\mathcal{U}\, \neg\varphi)$ for any LTL formulas ψ and φ.

exists a set of moves of the environment such that the system cannot satisfy φ. The knowledge of such initial state is useful since it provides the information about under what kind of conditions the system will fail to satisfy its desired property. The knowledge of the realizability of the specification is also useful in the development of complex systems. During the design process, the high level specification (also called the set of *requirements* by most system engineers) of the system is usually broken down into a set of specifications for different software modules. The designers of each software module are then responsible for developing their module such that its specification is satisfied. However, if the specification is not realizable, such a module cannot be developed. Without formally checking the realizability of the specification before the development, the software designers may waste a lot of time keeping implementing, testing and fixing their module until they finally realize that the specification they are given is impossible to be satisfied. Formally checking the realizability early in the design process allows the designer to adjust the specification of each module properly early in the design process which potentially save a lot of development time and cost.

If the specification is realizable, the synthesis algorithm generates a finite state automaton which represents a set of transitions the system should follow in order to satisfy φ. Assuming that the environment and the initial state of the system satisfy φ_e, at any instance of time, there exists a node in the automaton which represents the current state of the system, and the system can follow the transition from this node to the next based on the current knowledge about the environment. However, if φ_e is violated, the automaton is no longer valid, meaning that there may not exist a node in the automaton which represents the current state of the system, or even though such a node exists and the system follows the transitions in the automaton, the correct behavior φ_s is not guaranteed.

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. 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. In general, a dynamical system has a continuous domain and therefore $dom(V)$ is not finite. However, the synthesis algorithm described in Section II requires a finite domain. Thus, we partition $dom(S)$ and $dom(E)$ into a finite number of equivalence classes or cells $\mathcal{S}$ and $\mathcal{E}$, respectively, such that the partition is propositional preserving. 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 satisfy an atomic proposition π, denoted by $\nu \Vdash_d \pi$, if and only if there exists a continuous state v contained in the cell corresponding to the discrete state ν such that $v \Vdash \pi$. (Due to the propositional preserving property of the partition, this implies that for all continuous state v contained in the cell corresponding to the discrete state ν, $v \Vdash \pi$.) Given an infinite sequence of discrete state $\sigma_d = \nu_0\nu_1\nu_2\ldots$ and an LTL formula φ of variables from V, we say that φ holds at position $i \geq 0$ of σ_d, written $\nu_i \vDash_d \varphi$ if and only if φ holds for the remainder of σ_d. With these definitions, the semantics of LTL for a sequence of discrete state can be derived from the general semantics of LTL defined in Section II-B.

Assume that the value of $s \in S$ evolves according to the following piecewise-affine (PWA) dynamics[2]:

$$\begin{array}{rcl} s[t+1] & = & A_k s[t] + B_k u[t] + C_k \text{ if } (s[t], u[t]) \in D_k \\ u[t] & \in & U \end{array} \quad (4)$$

where $k \in \{1, \ldots, N_{PWA}\}$, N_{PWA} is the total number of PWA dynamics defined over a polyhedral partition of $dom(S) \times U$, for any natural number t, $s[t] \in dom(S)$ is the continuous controlled state at time t, u is the control signal and U is the set of admissible control inputs. Given a model of the system (4) and an LTL specification φ of variables from V, assuming that φ does not contain the next ($\bigcirc$) operator, we are interested in designing a discrete planner and a continuous controller for the system which ensure that any execution $\sigma = v_0 v_1 \ldots$ satisfies φ where for each natural number t, $v_t \in dom(V)$ is the continuous state of the system at time t.

IV. STATE SPACE DISCRETIZATION

Based on the partition described in Section III, we follow the scheme in [9] to obtain a corresponding finite transition system $\mathbb{D}$ which serves as an abstract model of $\mathbb{S}$. However, in our case, the dynamics (4), in general, constrains the evolution of the continuous state of the system which subsequently constrains a set of valid transitions of $\mathbb{D}$. As a result, the transition relation of $\mathbb{D}$ cannot be simply defined based on the topological adjacencies of the cells as in [9]. Furthermore, constructing a bisimulation partition for a general system with PWA dynamics is hard. In this section, we relax the requirement that the partition is bisimulation and define the notion of *reachability* which 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* by a continuous controller. This means that starting from any state $s[0] \in \mathcal{S}_i$, the continuous controller can drive the system to a state $s[T] \in \mathcal{S}_j$ satisfying the constraint $\forall t \in \{0, \ldots, T\}, s[t] \in \mathcal{S}_i \cup \mathcal{S}_j$.[3] A computational scheme which provides sufficient condition for reachability between two discrete controlled state is also presented.

A. Reachability Relationship

Let $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ be a partition of $dom(S)$ as described in Section III, i.e., for each $i, j \in \{1, \ldots, n\}$, $\mathcal{S}_i \subseteq$

[2]We restrict ourselves to PWA dynamics for computational reasons. Our framework applies to nonlinear dynamics but no computational scheme to date is able to handle them efficiently.

[3]By the abuse of notation, we use $\mathcal{S}_i$ and $\mathcal{S}_j$ to represent both the discrete states and the cells in the partition corresponding to those discrete states.

$dom(S)$, $int(\mathcal{S}_i) \cap int(\mathcal{S}_j) = \varnothing$ and $\bigcup_{i=1}^{n} \mathcal{S}_i = dom(S)$. We define the reachability relationship, 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 $s_{init} \in \mathcal{S}_i$, there exists a control law $u \in U$ which takes the system (4) to a point $s_{final} \in \mathcal{S}_j$ while always staying in $\mathcal{S}_i \cup \mathcal{S}_j$. Note that this is stronger than the usual definition of reachability [14]. We write $\mathcal{S}_i \not\rightsquigarrow \mathcal{S}_j$ if $\mathcal{S}_i$ is not reachable from $\mathcal{S}_j$. Clearly, if $\mathcal{S}_i \cap \mathcal{S}_j = \varnothing$ (i.e. $\mathcal{S}_i$ and $\mathcal{S}_j$ are not topologically adjacent), then $\mathcal{S}_i \not\rightsquigarrow \mathcal{S}_j$.

In general, for two discrete states $\mathcal{S}_i$ and $\mathcal{S}_j$, verifying the reachability relationship $\mathcal{S}_i \rightsquigarrow \mathcal{S}_j$ is hard. Therefore, we resort to a heuristic based on the following optimal control problem: Given cells $\mathcal{S}_i, \mathcal{S}_j \subseteq dom(S)$, the set of admissible control inputs U, the matrices A_k and B_k as in (4), a horizon length $N \geq 0$ and the cost matrices $P_N, Q \succeq 0$ and $R \succ 0$, solve

$$\begin{aligned} \min_{u[0],\ldots,u[N-1]} \quad & \|P_N s[N]\|_2 + \sum_{t=0}^{N-1} \|Qs[t]\|_2 + \|Ru[t]\|_2 \\ \text{s.t.} \quad & s[N] \in \mathcal{S}_j, \quad s[0] \in dom(S) \\ & s[t+1] = A_k s[t] + B_k u[t] \text{ if } (s[t], u[t]) \in D_k \\ & u[t] \in U \\ & s[t] \in \mathcal{S}_i \cup \mathcal{S}_j \\ & \forall t \in \{0, \ldots, N-1\}. \end{aligned} \tag{5}$$

Note that (5) is a finite horizon optimal control problem. Furthermore, one can consider the problem in (5) as a family of problems parametrized by $s[0]$ and it can be regarded as a multiparametric programming problem [3]. For certain choices of $\mathcal{S}_i$, $\mathcal{S}_j$, and U (for example polytopic sets, i.e., sets defined by affine inequalities), the Multi-Parametric Toolbox [4] computes the explicit solution for this multiparametric programming problem, i.e., computes a partition $\mathcal{P}_{i,j}$ of some subset of $\mathcal{S}_i \cup \mathcal{S}_j$ such that for any $s[0]$ contained in $\mathcal{P}_{i,j}$ the problem in (5) is feasible. 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 (5) is infeasible. Different unshaded regions have different associated controllers. For more detail, see [4].

B. State Space Discretization

In general, given the original partition $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ 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 multiparametric programming problem (5) since $\mathcal{S}_i$ 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 relationship 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 (5) is feasible. Then, we partition $\mathcal{S}_i$ into $\mathcal{S}_i \cap \mathcal{P}_{i,j}$ and $\mathcal{S}_i \setminus \mathcal{P}_{i,j}$ and obtain the following reachability relationship: $(\mathcal{S}_i \cap \mathcal{P}_{i,j}) \rightsquigarrow \mathcal{S}_j$ and $(\mathcal{S}_i \setminus \mathcal{P}_{i,j}) \not\rightsquigarrow \mathcal{S}_j$.

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[4]. Starting with a pair (i, j) where $i, j \in \{1, \ldots, n\}$ and $i \neq j$, determine the set $\mathcal{P}_{i,j}$ such that for any $s[0] \in \mathcal{P}_{i,j}$, the problem in (5) is feasible. Partition $\mathcal{S}_i$ into $\mathcal{S}_i \cap \mathcal{P}_{i,j}$ and $\mathcal{S}_i \setminus \mathcal{P}_{i,j}$ and replace $\mathcal{S}_i$ in $\mathcal{S}$ with $\mathcal{S}_i \cap \mathcal{P}_{i,j}$ and $\mathcal{S}_i \setminus \mathcal{P}_{i,j}$ if the volumes of both $\mathcal{S}_i \cap \mathcal{P}_{i,j}$ and $\mathcal{S}_i \setminus \mathcal{P}_{i,j}$ are greater than Vol_{min}. Keep iterating this process until none of the cells in $\mathcal{S}$ can be partitioned.

We let $\mathcal{S}' = \{\mathcal{S}'_1, \mathcal{S}'_2, \ldots, \mathcal{S}'_m\}$ be the resulting partition of $dom(S)$ after applying the proposed discretization algorithm. Since $\mathcal{S}'$ is a subpartition of $\mathcal{S}$ and $\mathcal{V} = \mathcal{S} \times \mathcal{E}$ is propositional preserving, it is trivial to show that $\mathcal{V}' = \mathcal{S}' \times \mathcal{E}$ is also propositional preserving. As before, we can construct a corresponding finite transition system $\mathbb{D}'$ based on the partition $\mathcal{V}'$ of the continuous domain V.

Remark 1: 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 2: At which point the algorithm terminates affects the reachability between cells 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. Specification Modification

To restrict discrete transitions to those satisfying the reachability relation, we simply add extra LTL formulas of the form $\Box(\psi_1 \Longrightarrow \bigcirc\psi_2)$ to φ_s in the system specification (3). Given the new partition $\mathcal{S}' = \{\mathcal{S}'_1, \mathcal{S}'_2, \ldots, \mathcal{S}'_m\}$ of $dom(S)$, we determine the reachability between $\mathcal{S}'_i$ and $\mathcal{S}'_j$ for $i, j \in \{1, \ldots, m\}$.

For each $i \in \{1, \ldots, m\}$, let $\mathcal{S}_i^{reach}$ be a subset of $\mathcal{S}'$ such that for any $\mathcal{S}'_j \in \mathcal{S}_i^{reach}$, $\mathcal{S}'_i \rightsquigarrow \mathcal{S}'_j$. For each i such that $\mathcal{S}_i^{reach}$ is not empty, we add the following formula to φ_s:

$$\Box\left((s \in \mathcal{S}'_i) \implies \bigcirc\left(\bigvee_{\mathcal{S}'_j \in \mathcal{S}_i^{reach}} (s \in \mathcal{S}'_j)\right)\right). \tag{6}$$

For each i such that $\mathcal{S}_i^{reach}$ is empty, we add the formula $\Box(\neg(s \in \mathcal{S}'_i))$ to φ_s to make sure that the continuous controlled state will never lie in $\mathcal{S}'_i$ since there exists a point in $\mathcal{S}'_i$ from which the transition to other discrete state may not be achievable by the continuous controller.

[4] 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.

From the formation of the optimal control problem (5) and the propositional preserving property of the partition, it is straightforward to show the following proposition.

Proposition 1: Let φ be an LTL formula which does not contain the next ($\bigcirc$) operator. Suppose a sequence of discrete transition of $\mathbb{D}$ $\sigma_d = \nu_0\nu_1\ldots$ satisfy φ and for each $i \in \{0,1,\ldots\}$, $\nu_i \rightsquigarrow \nu_j$ and for all $s \in \nu_i$, (5) is feasible.

V. RECEDING HORIZON TEMPORAL LOGIC PLANNING

Automatic synthesis of a finite state automaton from its LTL specification [1] suffers from the state explosion problem. In many applications, however, the state that is very far from the current state of the system does not affect the determination of the near future plan. In this section, we present a receding horizon scheme which potentially substantially reduce the number of states of the automaton while still ensuring the correctness of the system.

Given an LTL formula φ, the discrete domain of the system $\mathcal{V} = \mathcal{S} \times \mathcal{E}$ and a subset $\mathcal{Z} \subseteq \mathcal{S}$, we let $\varphi\Big|_{\mathcal{Z}}$ represent the simplification of φ with restricted domain $\mathcal{Z} \times \mathcal{E}$ by assuming that the discrete controlled state of the system is always contained in $\mathcal{Z}$. That is, $\varphi\Big|_{\mathcal{Z}}$ is obtained from φ by replacing any of its atomic propositions of a variable from S which is assigned the value $True$ when the discrete control state $s \notin \mathcal{Z}$ with $False$. In addition, let $\varphi\Big\rceil_{\mathcal{Z}}$ represent the simplification of φ with restricted domain $\mathcal{Z} \times \mathcal{E}$ which is obtained from φ by replacing any of its atomic propositions of a variable from S which is assigned the value $True$ when $s \notin \mathcal{Z}$ with $True$ or $False$ such that $\varphi \implies \varphi\Big\rceil_{\mathcal{Z}}$ is a tautology, i.e. any discrete state $v \in \mathcal{V}$ that satisfies φ also satisfies $\varphi\Big\rceil_{\mathcal{Z}}$. For example, suppose $\varphi = \Box((s = \mathcal{S}_1) \vee (s = \mathcal{S}_2) \vee (s = \mathcal{S}_3))$ and $\mathcal{Z} = \{\mathcal{S}_1, \mathcal{S}_2\}$. Then, $\varphi\Big|_{\mathcal{Z}}$ is obtained by replacing the atomic formula $(s = \mathcal{S}_3)$ with $False$, so we get $\varphi\Big|_{\mathcal{Z}} = \Box((s = \mathcal{S}_1) \vee (s = \mathcal{S}_2))$. To obtain $\varphi\Big\rceil_{\mathcal{Z}}$, we see that to make $\varphi \implies \varphi\Big\rceil_{\mathcal{Z}}$ a tautology, we need to replace the atomic formula $(s = \mathcal{S}_3)$ with $True$, so we get $\varphi\Big\rceil_{\mathcal{Z}} = True$.

In this paper, we are interested in a subclass of generalized Reactivity(1) formulas (2). Let (a) $\mathcal{S}_{init} \subseteq \mathcal{S}$, (b) φ_{init} be a propositional formula of variables from V which characterizes the initial state of the system (c) φ_e be a boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_e$ where ψ_e is a propositional formula of variables from E which describes the assumptions on the transitions of environment states. (d) for each $j \in J$ where J is a finite set, φ_j be a propositional formula of variables from V, (e) φ_s be a boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_s$ where ψ_s is a propositional formula of variables from V which describes the constraints on the transitions of system states, and (f) φ_g be a propositional formula of variables from V. We consider a specification of the form

$$\begin{array}{c}\big((s \in \mathcal{S}_{init}) \wedge \varphi_{init} \wedge \Box\varphi_e \wedge \bigwedge_{j\in J}\Box\Diamond\varphi_j\big) \\ \implies \big(\Box\varphi_s \wedge \Diamond\varphi_g\big)\end{array} \tag{7}$$

For the progress property $\Diamond\varphi_g$, suppose there exists a collection of disjoint subsets $\mathcal{V}_1, \ldots, \mathcal{V}_p$ of $\mathcal{V}$ such that $\mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_p = \mathcal{V}$, φ_g is satisfied for any $v \in \mathcal{V}_p$, and $(\{\mathcal{V}_1, \ldots, \mathcal{V}_p\}, \preceq_{\varphi_g})$ is a lattice where for any i and j, $\mathcal{V}_i \preceq_{\varphi_g} \mathcal{V}_j$ if and only if any execution starting from $v \in \mathcal{V}_i$ and satisfying the property $\Diamond(v \in \mathcal{V}_p)$ contains a state $v \in \mathcal{V}_j$. Clearly, by the definition of $\preceq_{\varphi_g}$, for any $i \in \{1, \ldots, p\}$, $\mathcal{V}_i \preceq_{\varphi_g} \mathcal{V}_p$.

Suppose there exists a propositional formula Φ of variables from V such that $\varphi_{init} \implies \Phi$ is a tautology. Suppose further that for each $i \in \{1, \ldots, p\}$, there exist subsets $\mathcal{D}_i$ and $\mathcal{S}^i_{init}$ of $\mathcal{S}$ with

- $(\mathcal{V}_i \restriction S) \subseteq \mathcal{S}^i_{init} \subseteq \mathcal{D}_i$,
- there exists $g_i \in \{1, \ldots, p\}$ where $\mathcal{V}_i \preceq_{\varphi_g} \mathcal{V}_{g_i}$ and for each $i \neq p$, $\mathcal{V}_i \prec_{\varphi_g} \mathcal{V}_{g_i}$ such that $(\mathcal{V}_{g_i} \restriction S) \subseteq \mathcal{D}_i$

such that

$$\begin{array}{rcl}\Psi_i & \equiv & \Big((s \in \mathcal{S}^i_{init}) \wedge \Phi\Big|_{\mathcal{S}^i_{init}} \wedge \Box(\varphi_e\Big\rceil_{\mathcal{D}_i}) \wedge \\ & & \bigwedge_{j\in J}\Box\Diamond(\varphi_j\Big\rceil_{\mathcal{D}_i})\Big) \implies \\ & & \Big(\Box(\varphi_s\Big\rceil_{\mathcal{D}_i}) \wedge \Diamond(v \in \mathcal{V}_{g_i}) \wedge \Box(\Phi\Big\rceil_{\mathcal{D}_i})\Big)\end{array} \tag{8}$$

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

For $i \in \{1, \ldots, p\}$, let $\mathcal{A}_i$ be an automaton which 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 $\mathcal{S}$.

Receding Horizon Strategy: Starting from the state v_0, pick an automaton $\mathcal{A}_i$ such that $v_0 \in \mathcal{V}_i$ and execute $\mathcal{A}_i$ until the system reaches the state $v \in \mathcal{V}_j$ where $\mathcal{V}_j \succ_{\varphi_g} \mathcal{V}_i$, at which point, switch to the automaton $\mathcal{A}_j$. Keep iterating this process until $\mathcal{A}_p$ is executed.

Theorem 1: Suppose for each $i \in \{1, \ldots, p\}$, 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 (7). From the definition of $\mathcal{S}^i_{init}$, $\Phi\Big|_{\mathcal{S}^i_{init}}$, $\varphi_e\Big\rceil_{\mathcal{D}_i}$ and $\varphi_j\Big\rceil_{\mathcal{D}_i}$, it is easy to show that if σ starts from $v \in \mathcal{V}_i$ such that v satisfies Φ, then σ satisfies the formula to the left of $\implies$ in (8). Let $v_0 \in \mathcal{V}$ be the initial state of the system. First, suppose $v_0 \in \mathcal{V}_p$. Then, the system always executes $\mathcal{A}_p$ and since $\varphi_{init} \implies \Phi$ is a tautology, Ψ_p ensures that σ satisfies (7). Next, suppose $v_0 \in \mathcal{V}_i$ where $i \neq p$. Then, $v_0 \restriction S \in \mathcal{S}^i_{init}$ and Ψ_i ensures that the safety property φ_s holds at all the states in σ up to and including the state at which the system switches the automaton and at this state, Φ is satisfied. In addition, from the definition of $\preceq_{\varphi_g}$, Ψ_i ensures that eventually the system reaches the state $v_j \in \mathcal{V}_j$ where $\mathcal{V}_i \prec_{\varphi_g} \mathcal{V}_j$. According to the receding horizon scheme, the system switch an automaton at this state. Since $v_j \in \mathcal{V}_j$ and v_j satisfies Φ, from the previous argument, σ satisfies the

formula to the left of $\Longrightarrow$ in (8). By the same argument as for v_0, Ψ_j ensures that the safety property φ_s holds at all the states in σ after the system switches to $\mathcal{A}_j$ and up to and including the state at which the system switches the automaton and at this state, Φ is satisfied. Keep iterating this proof, we get that φ_s holds at all the states in σ and due to the finiteness of the set $\{\mathcal{V}_1, \ldots, \mathcal{V}_p\}$ and its partial order structure, eventually the automaton $\mathcal{A}_p$ is executed which ensures that σ satisfies the progress property $\Diamond\varphi_g$. ■

Remark 3: Roughly speaking, identifying the partial order structure of $\mathcal{V}$ is similar to finding a control Lyapunov function in the receding horizon control (RHC) framework. In RHC, a control Lyapunov function provides a partial order structure for a special case of the system whose evolution is modeled by a differential equation and whose state lies in a normed real vector space. In our temporal logic planning framework, the evolution of the system is modeled by the transitions between discrete states in the automaton. The cardinality of the chain $(\mathcal{V}_i, \mathcal{V}_{g_i})$ in the partial order structure can be viewed as the horizon length for the automaton $\mathcal{A}_i$.

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 and is used to make Ψ_i realizable. One way to determine Φ is to start with $\Phi = True$ and check the realizability of the resulting Ψ_i. If for any $i \in \{1, \ldots, p\}$, Ψ_i is realizable, we are done. Otherwise, the synthesis process provides the initial state of the system for 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.

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 [15] that the nondimensional equations of motion of the vehicle is given by

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} \dot{x} \\ \dot{y} \\ \frac{2mL^2}{J}\dot{\theta} \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \\ q_\theta \end{bmatrix} \tag{9}$$

with the following constraints on the control efforts

$$\forall t, q_x^2(t) + q_y^2(t) \le \left(\frac{3 - |q_\theta(t)|}{2}\right)^2 \text{ and } |q_\theta(t)| \le 3. \tag{10}$$

Conservatively, we can set $|q_x(t)| \le \sqrt{0.5}$, $|q_y(t)| \le \sqrt{0.5}$ and $|q_\theta(t)| \le 1$ so that the constraints (10) are decoupled.

In this section, we are only interested in the translational (x and y) components of the vehicle state. Discretizing the dynamics (9) with time step 0.1, we obtain the following discrete-time linear time-invariant state space model

$$\begin{bmatrix} z[t+1] \\ v_z[t+1] \end{bmatrix} = \begin{bmatrix} 1 & 0.0952 \\ 0 & 0.9048 \end{bmatrix} \begin{bmatrix} z[t] \\ v_z[t] \end{bmatrix} + \begin{bmatrix} 0.0048 \\ 0.0952 \end{bmatrix} q_z \tag{11}$$

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

$$C_z = \bigcup_{i \in \{zmin+1, \ldots, zmax\}} C_{z,i} \tag{12}$$

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.

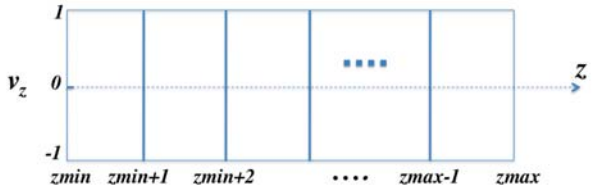

Fig. 2. The original partition of the domain C_z

We consider the road with 2 lanes, each of width 1, so we set $ymin = 0$ and $ymax = 2$. Since the vehicle dynamics are translational 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 $O_{i,j}$ which 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 $(O_{1,1}, O_{1,2}, \ldots, O_{L,1}, O_{L,2}) \in \{0, 1\}^{2L}$ 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, φ_{init} in (7) is defined as for any $i \in \{1 + d_{obs}, \ldots, L - d_{obs}\}$,

$$\left(x \in \bigcup_{k=i-d_{obs}}^{i+d_{obs}} C_{x,k} \Longrightarrow (\neg O_{i,1} \wedge \neg O_{i,2})\right) \wedge y \in C_{y,1}. \tag{13}$$

The following properties are assumed for the environment.

1) An obstacle is always detected before the vehicle gets too close to it. That is, there is a lower bound d_{popup} 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

$$\Box\left(\left(x \in \bigcup_{j=i-d_{popup}}^{i+d_{popup}} C_{x,j} \wedge \neg O_{i,k}\right) \Longrightarrow \bigcirc(\neg O_{i,k})\right) \tag{14}$$

for all $i \in \{1 + d_{popup}, \ldots, L - d_{popup}\}$ and $k \in \{1, 2\}$.

2) Sensing range is limited. That is, the vehicle cannot detect an obstacle that is away from it farther than d_{sr}. An LTL formula corresponding to this assumption is a conjunction of the following formula

$$\Box\left(x \in C_{x,i} \Longrightarrow \bigwedge_{j > i+d_{sr}} (\neg O_{j,1} \wedge \neg O_{j,2})\right) \tag{15}$$

for all $i \in \{1, \ldots, L\}$.

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

$$\Box\left(\neg O_{i,1} \vee \neg O_{i,2}\right) \tag{16}$$

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. That is, for any $i \in \{1, \ldots, L\}$,

$$\Box\left(O_{i,1} \implies \bigcirc(O_{i,1})\right) \tag{17}$$

We define $\Box\varphi_e$ in (7) to be the conjunction of formula (14)-(17). (Note that for any LTL formulas φ and ψ, $\Box\varphi \wedge \Box\psi$ is equivalent to $\Box(\varphi \wedge \psi)$.)

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

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

$$\Box(O_{i,j} \implies \neg(x \in C_{x,i} \wedge y \in C_{y,j})) \tag{18}$$

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

$$\Box((\neg O_{i,1} \wedge x \in C_{x,i}) \implies (y \in C_{y,1})) \tag{19}$$

Finally, we define $\varphi_g = (x \in C_{x,L})$. That is, we want to guarantee 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[5]. Since the vehicle dynamics (9) are translational invariant, we can use similar partition for all $C_{z,i}$. The discretization algorithm with horizon length $N = 10$ and $Vol_{min} = 0.1$ yields 11 partitions for each $C_{z,i}$ as shown in Fig. 3. Using MPT [4], the reachability between different cells in the new partition can be determined and a set of atomic controllers associated with a pair of cells can be generated such that the bisimulation property is satisfied. The original specification is then modified as discussed in Section IV-C.

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 the state explosion problem, we apply the receding horizon strategy on the variable x. For $i \in \{zmin+1, \ldots, zmax\}$, let $\mathcal{C}_{z,i} = \{\mathcal{C}^1_{z,i}, \mathcal{C}^2_{z,i}, \ldots, \mathcal{C}^{11}_{z,i}\}$ be the partition of $C_{z,i}$. The partial order structure is defined as $\mathcal{V}_i = \{(x, v_x, y, v_y, O_{1,1}, \ldots, O_{L,2}) \mid x \in \{\mathcal{C}^1_{z,i}, \mathcal{C}^2_{z,i}, \ldots, \mathcal{C}^{11}_{z,i}\}\}$. It can be easily shown that for any $i < j$, $\mathcal{V}_i \prec_{\varphi_g} \mathcal{V}_j$ since to get to the cell $C_{x,L}$ starting from $C_{x,i}$, the system needs to visit a cell $C_{x,j}$ for any $j > i$.

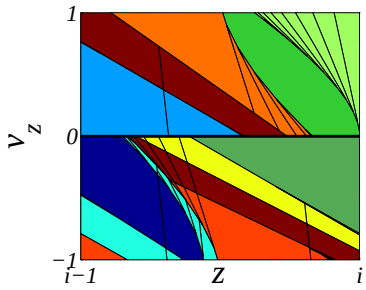

Fig. 3. The partition of each cell $C_{z,i}$ in the original partition of the domain C_z

Next, we need to define an invariant Φ such that the specification (7) is realizable. Similar to z, we let $\mathcal{Z}$ represent either $\mathcal{X}$ or $\mathcal{Y}$. We classify the cells in the new partition into one of the following catagories:

1) A set $\mathcal{Z}_{notr}$ of cells from which the system cannot transition to any cells. That is, for any $\mathcal{C} \in \mathcal{Z}_{notr}$, $i \in \{zmin+1, \ldots, zmax\}$ and $j \in \{1, \ldots, 11\}$, $\mathcal{C} \nrightarrow \mathcal{C}^j_{z,i}$.
2) A set $\mathcal{Z}_{right}$ of cells from which the system can only transition to a cell to the right of it in the original partition. That is, for any $\mathcal{C} \in \mathcal{Z}_{right}$ such that $\mathcal{C} \subseteq C_{z,i}$, there exists $k \in \{1, \ldots, 11\}$ such that $\mathcal{C} \rightsquigarrow \mathcal{C}^k_{z,i+1}$. In addition, for any $j \neq i+1$ and $k \in \{1, \ldots, 11\}$, $\mathcal{C} \nrightarrow \mathcal{C}^k_{z,j}$.
3) A set $\mathcal{Z}_{left}$ of cells from which the system can only transition to a cell to the left of it in the original partition. That is, for any $\mathcal{C} \in \mathcal{Z}_{left}$ such that $\mathcal{C} \subseteq C_{z,i}$, there exists $k \in \{1, \ldots, 11\}$ such that $\mathcal{C} \rightsquigarrow \mathcal{C}^{k+}_{z,i-1}$. In addition, for any $j \neq i-1$ and $k \in \{1, \ldots, 11\}$, $\mathcal{C} \nrightarrow \mathcal{C}^k_{z,j}$.
4) A set $\mathcal{Z}_{same}$ of cells from which the system can only transition to the same cell in the original partition. That is, for any $\mathcal{C} \in \mathcal{Z}_{same}$ such that $\mathcal{C} \subseteq C_{z,i}$, there exists $k \in \{1, \ldots, 11\}$ such that $\mathcal{C} \rightsquigarrow \mathcal{C}^k_{z,i}$. In addition, for any $j \neq i$ and $k \in \{1, \ldots, 11\}$, $\mathcal{C} \nrightarrow \mathcal{C}^k_{z,j}$.
5) A set $\mathcal{Z}_{both}$ of cells from which the system can transition to both the same cell in the original partition and a different cell in the original partition. That is, for any $\mathcal{C} \in \mathcal{Z}_{both}$ such that $\mathcal{C} \subseteq C_{z,i}$, there exists $k \in \{1, \ldots, 11\}$ such that $\mathcal{C} \rightsquigarrow \mathcal{C}^k_{z,i}$. In addition, there exists $j \neq i$ and $k \in \{1, \ldots, 11\}$ such that $\mathcal{C} \rightsquigarrow \mathcal{C}^k_{z,j}$.

To restrict the initial states of the system assumed by Ψ_i so that Ψ_i is realizable, we make the following observations.

1) To ensure the progress property $\diamond\varphi_g$, x cannot be in a cell $\mathcal{C} \in \mathcal{X}_{notr}$ and y cannot be in a cell $\mathcal{C} \in \mathcal{Y}_{notr}$.
2) To ensure no collision, the vehicle cannot collide with an obstacle at the initial state.
3) Suppose x is in the cell $C_{x,i}$ in the original partition. To ensure no collision, if y can only transition to the cell $C_{y,1}$ in the original partition, then either $O_{i,1}$ is *False* or $O_{i+1,1}$ is *False*. Similarly, if y can only transition to

[5]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.

the cell $C_{y,2}$ in the original partition, then either $O_{i,2}$ is $False$ or $O_{i+1,2}$ is $False$.

4) Suppose x is in the cell $C_{x,i}$ in the original partition such that it can only transition to the cell to the right of it in the original partition. To ensure no collision, if y can only transition to the cell $C_{y,1}$ in the original partition, then $O_{i+1,1}$ is $False$. Similarly, if y can only transition to the cell $C_{y,2}$ in the original partition, then $O_{i+1,2}$ is $False$.
5) Suppose x is in the cell $C_{x,i}$ in the original partition such that it can only transition to the same cell $C_{x,i}$ in the original partition. To ensure no collision, if y is in the cell $C_{y,2}$ and can only transition to a cell $C_{y,1}$ in the original partition, then $O_{i,1}$ is $False$. Similarly, if y is in the cell $C_{y,1}$ and can only transition to a cell $C_{y,2}$ in the original partition, then $O_{i,2}$ is $False$.
6) 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 where x can only transition to the same cell in the original partition and y is in the cell $C_{y,1}$ in the original partition and can only transition to the cell $C_{y,2}$ in the original partition.
7) Suppose x is in the cell $C_{x,i}$ in the original partition 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 y is in the cell $C_{y,1}$ in the original partition and y can only transition to the cell $C_{y,2}$ in the original partition. In addition, it is not the case that x can only transition to the cells $C_{x,i+1}$ in the original partition and y is in the cell $C_{y,2}$ in the original partition can only transition to the same cell in the original partition.

Based on the above observations, we define Φ to be the conjunction of the following formulas:

1) $\neg(\bigvee_{\mathcal{C}\in\mathcal{X}_{notr}} x \in \mathcal{C}) \wedge \neg(\bigvee_{\mathcal{C}\in\mathcal{Y}_{notr}} y \in \mathcal{C})$
2) $O_{i,j} \implies \neg(x \in C_{x,i} \wedge y \in C_{y,j})$
3) $((((\bigvee_{\mathcal{C}\in\mathcal{Y}_{same}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,1})) \vee ((\bigvee_{\mathcal{C}\in\mathcal{Y}_{left}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,2}))) \wedge (\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i})) \implies (\neg O_{i,1} \vee \neg O_{i+1,1})$
4) $((((\bigvee_{\mathcal{C}\in\mathcal{Y}_{right}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,1})) \vee ((\bigvee_{\mathcal{C}\in\mathcal{Y}_{same}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,2}))) \wedge (\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i})) \implies (\neg O_{i,2} \vee \neg O_{i+1,2})$
5) $((((\bigvee_{\mathcal{C}\in\mathcal{Y}_{same}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,1})) \vee ((\bigvee_{\mathcal{C}\in\mathcal{Y}_{left}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,2}))) \wedge ((\bigvee_{\mathcal{C}\in\mathcal{X}_{right}} x \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i}))) \implies \neg O_{i+1,1}$
6) $((((\bigvee_{\mathcal{C}\in\mathcal{Y}_{right}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,1})) \vee ((\bigvee_{\mathcal{C}\in\mathcal{Y}_{same}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,2}))) \wedge ((\bigvee_{\mathcal{C}\in\mathcal{X}_{right}} x \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i}))) \implies \neg O_{i+1,2}$
7) $(((\bigvee_{\mathcal{C}\in\mathcal{X}_{same}} x \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i})) \wedge ((\bigvee_{\mathcal{C}\in\mathcal{Y}_{left}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,2}))) \implies \neg O_{i,1}$
8) $(((\bigvee_{\mathcal{C}\in\mathcal{X}_{same}} x \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i})) \wedge ((\bigvee_{\mathcal{C}\in\mathcal{Y}_{right}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,1}))) \implies \neg O_{i,2}$
9) $((\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,2})) \implies O_{i,1}$
10) $\neg(((\bigvee_{\mathcal{C}\in\mathcal{X}_{same}} x \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i})) \wedge ((\bigvee_{\mathcal{C}\in\mathcal{Y}_{right}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,1})))$
11) $\neg Obs_{i+1,1} \implies \neg(((\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i}) \wedge (((\bigvee_{\mathcal{C}\in\mathcal{Y}_{right}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^j_{y,1})) \vee (((\bigvee_{\mathcal{C}\in\mathcal{X}_{right}} x \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} x \in \mathcal{C}^j_{x,i})) \wedge ((\bigvee_{\mathcal{C}\in\mathcal{Y}_{same}} y \in \mathcal{C}) \wedge (\bigvee_{j\in\{1,\dots,11\}} y \in \mathcal{C}^{j+}_{y,2})))))$

The first formula constrains the state of the system to ensure that the progress property is satisfied (observation 1). The next seven formulas constrain the state of the system to ensure that the obstacle avoidance property is satisfied (observations 2-5) and the last three formulas constrain the state of the system to ensure that the stay-in-lane property is satisfied (observations 6-7).

With $d_{popup} = 1$ and the horizon length 2 (i.e. $g_i = i + 2$), the specification (8) is realizable. In addition, if we let d_{obs} be greater than 1 and restrict the initial state of the system such that x is not in a cell $\mathcal{C} \in \mathcal{X}_{notr}$ and y is not in a cell $\mathcal{Y}_{notr}$, we get that $\varphi_{init} \implies \Phi$.

D. Results

The synthesis was performed on a Pentium 4, 3.4 MHz 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.

A simulation result with the road length of 30 is shown in Fig. 4. The polygons drawn in red are the obstacles. 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.

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 which are guaranteed, by construction, to satisfy the desired properties expressed in temporal logic even in the presence of adversary. A receding horizon scheme was described which addresses the main limitation of the synthesis algorithm, the state explosion problem, and allows more complex problems to be solved, assuming that the system has a lattice structure. The example showed that without the receding horizon scheme,

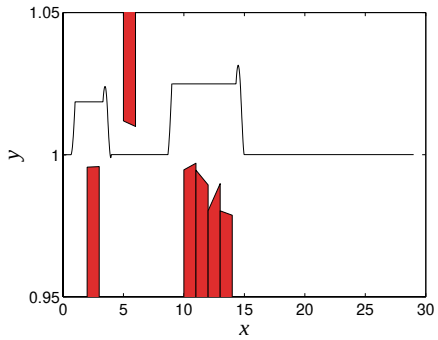

Fig. 4. Simulation result. The polygons drawn in red are the obstacles.

the synthesis problem can be extremely computationally challenging.

Although the adversary nature of the environment has been incorporated in the synthesis, the effects of disturbances and modelling errors have not yet been studied. To increase the robustness of the system, we plan to impose more conditions on the multiparametric 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 modelling 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 the specification is realizable, a propositional formula which describes the initial state of the system for 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.

REFERENCES

[1] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *7th International Conference on Verification, Model Checking and Abstract Interpretation*, vol. 3855 of *Lecture Notes in Computer Science*, pp. 364 – 380, Springer-Verlag, 2006.

[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] 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.

[8] 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.

[9] G. Fainekos, H. Kress-Gazit, and G. Pappas, "Temporal logic motion planning for mobile robots," *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 2020–2025, April 2005.

[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, 29 2007-Nov. 2 2007.

[12] 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.

[13] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2 ed., 2004.

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

[15] 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.