Synthesis of Provably Correct Controllers for Autonomous Vehicles in Urban Environments

Tichakorn Wongpiromsarn, Sertac Karaman and Emilio Frazzoli

***Abstract*— This paper considers automatic synthesis of provably correct controllers for autonomous vehicles operating in an urban environment populated with static obstacles and live traffic. We express traffic rules such as collision avoidance, vehicle separation, speed limit, lane following, passing, merging and intersection precedence requirements in a formal specification language. Embedded control software synthesis is then applied to generate a controller that ensures that the vehicle obeys this set of traffic rules in any road and traffic conditions that satisfy certain assumptions.**

I. INTRODUCTION

In the past decades, there have been growing interests in increasing the level of autonomy in transportation systems to enhance operational safety and efficiency. In particular, considerable research has been directed towards putting more autonomy into transport vehicles. The vehicles completed the 2007 DARPA Urban Challenge and the Google driverless cars have demonstrated that the technology for autonomous vehicles is close. However, to allow these vehicles to drive on a road with other autonomous and human-driven vehicles, it is crucial to verify that the autonomous vehicles operate safely and correctly follow traffic rules in any situation. As the number of possible situations is large, such verification can be a challenging task. Verification of certain components can be found, for example, in [1], [2].

In this paper, we consider an alternative approach where a "correct" controller is automatically designed such that, by construction, the vehicle is guaranteed to obey traffic rules in any traffic conditions that satisfy certain assumptions. To enable such an automatic construction, we precisely describe the vehicle specification in a formal language. The vehicle specification incorporates assumptions on the road and traffic conditions as well as traffic rules including collision avoidance, vehicle separation, speed limit, lane following, passing, merging and intersection precedence requirements. We use linear temporal logic [3] as a specification language due to its expressiveness. Embedded control software synthesis is then applied to automatically construct a controller that is correct with respect to the vehicle specification.

Automatic construction of a controller for an autonomous vehicle has been considered, for example, in [4], [5], [6], [7] with a limited set of traffic rules. The main contribution of this paper is in extending the set of traffic rules to cover all the nominal road driving requirements in the technical evaluation criteria of the DARPA Urban Challenge. As a result, the autonomous vehicle exhibits much more similar behaviors to those of human-driven vehicles. For example, the vehicle respects the precedence order at an intersection rather than having to wait until the intersection is clear. It also stops within a required distance and monitors oncoming traffic before performing a passing maneuver. To achieve this, a finer abstraction that that used in, e.g., [4], [5] is required and results in additional computational complexity. The receding horizon framework proposed in [6], [7] is applied in order to keep the problem manageable.

The remainder of the paper is organized as follows. In Section II, we briefly describe linear temporal logic and terminology and notations used throughout the paper. Section III formulates the urban autonomous driving problem as well as formally describes the vehicle specification. Section IV provides a brief overview of existing techniques for embedded control software synthesis. Simulation results demonstrating that the autonomous vehicle correctly obeys traffic rules are presented in Section V.

II. PRELIMINARIES

We use linear temporal logic (LTL) to describe system specifications. LTL is a powerful specification language for unambiguously expressing a wide range of desired system behaviors and has been utilized in various applications [8].

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 $v \in dom(V)$.

In this paper, we consider the case where V can be partitioned into two disjoint subsets S and E of controlled and environment variables, respectively. The environment variables are those related to factors over which the system does not have control whereas the controlled variables are those whose values can be set by the controller.

Definition 2: An *atomic proposition* is a statement on system variables $v \in V$ that has a unique truth value ($True$ or $False$) for a given value of v. Let $\nu \in dom(V)$ be a state of the system and p be an atomic proposition. We write $\nu \Vdash p$ if p is $True$ at the state ν. Otherwise, we write $\nu \nVdash p$.

Definition 3: A *finite transition system* is a tuple $\mathbb{T} := (\mathcal{V}, U, \rightarrow, \mathcal{V}_0, \Pi, \mathcal{L})$ where $\mathcal{V}$ is a finite set of states, U is a finite set of actions, $\rightarrow \subseteq \mathcal{V} \times U \times \mathcal{V}$ is a transition relation, $\mathcal{V}_0 \subseteq \mathcal{V}$ is a set of initial states, Π is a set of atomic propositions, and $\mathcal{L} : \mathcal{V} \rightarrow 2^{\Pi}$ is a labeling function.

Definition 4: An *execution* σ of a discrete-time system is an infinite sequence of system states over a particular run, i.e., σ can be written as $\sigma = v_0 v_1 v_2 \ldots$ where for each $t \in \mathbb{N}_0$, $v_t = v[t] \in dom(V)$ is the system state at time t.

T. Wongpiromsarn is with the Singapore-MIT Alliance for Research and Technology, Singapore nok@smart.mit.edu

S. Karaman and E. Frazzoli are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA {sertac, frazzoli}@mit.edu

Linear temporal logic is built up from a set of atomic propositions, the logic connectives ($\neg$, $\vee$, $\wedge$, $\Longrightarrow$), and the temporal modal operators ($\bigcirc$, $\Box$, $\Diamond$, $\mathcal{U}$, which are read as "*next*," "*always*," "*eventually*," and "*until*," respectively). An LTL formula is defined inductively as follows: (1) any atomic proposition p is an LTL formula; and (2) given LTL formulas φ and ψ, $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$ and $\varphi\,\mathcal{U}\,\psi$ are also LTL formulas. Other operators can be defined as follows: $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$, $\varphi \Longrightarrow \psi \equiv \neg\varphi \vee \psi$, $\Diamond\varphi \equiv \mathit{True}\,\mathcal{U}\,\varphi$, and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. A *propositional* formula is one that does not include any temporal operator.

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 write $v_i \models \varphi$ if φ holds at position $i \in \mathbb{N}_0$ of σ. The semantics of LTL is defined inductively as follows: (a) For an atomic proposition p, $v_i \models p$ if and only if (iff) $v_i \Vdash p$; (b) $v_i \models \neg\varphi$ iff $v_i \not\models \varphi$; (c) $v_i \models \varphi \vee \psi$ iff $v_i \models \varphi$ or $v_i \models \psi$; (d) $v_i \models \bigcirc\varphi$ iff $v_{i+1} \models \varphi$; and (e) $v_i \models \varphi\,\mathcal{U}\,\psi$ iff there exists $j \geq i$ such that $v_j \models \psi$ and $\forall k \in [i, j), v_k \models \varphi$. Based on this definition, $\bigcirc\varphi$ holds at position v_i iff φ holds at the next state v_{i+1}, $\Box\varphi$ 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 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, $(\Sigma \models \varphi) \iff \big(\forall\sigma, (\sigma \in \Sigma) \implies (\sigma \models \varphi)\big)$.

Safety (invariance) and reachability are examples of widely used properties. Given a propositional formula p that describes states of interest, a safety formula is of the form $\Box p$, which asserts that p remains invariantly true throughout an execution. A reachability formula is of the form $\Diamond p$, which asserts that p becomes true at least once in an execution. Obstacle avoidance and reaching a goal state are examples of safety and reachability properties, respectively.

III. URBAN AUTONOMOUS DRIVING PROBLEM

We consider the problem of autonomous driving in partially known urban environments populated with static obstacles and live traffic similar to those faced in the 2007 DARPA Urban Challenge (DUC). The aim is to automatically synthesize a controller that ensures that the vehicle obeys traffic rules while executing a given driving task.

A. Problem Setup

A road network $\mathcal{R}$ is defined as a collection of lanes and intersections. A lane is connected to another via an intersection. Let N_l and N_{int} be the number of lanes and intersections, respectively, in $\mathcal{R}$. Then, $\mathcal{R}$ can be written as $\mathcal{R} = (\mathcal{L}, \mathcal{I})$ where $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_{N_l}\}$ and $\mathcal{I} = \{\mathcal{I}_1, \ldots, \mathcal{I}_{N_{int}}\}$ are sets of all the lanes and intersections, respectively, in $\mathcal{R}$. Without loss of generality, we assume that each lane can be exited through only one intersection.

A lane is defined as a tuple $\mathcal{L}_i := (X_{\mathcal{L}_i}, d_{\mathcal{L}_i}, v_{\mathcal{L}_i}, Entry_{\mathcal{L}_i}, Exit_{\mathcal{L}_i}, Stop_{\mathcal{L}_i}, Adj_{\mathcal{L}_i})$ where $i \in \{1, \ldots, N_l\}$, $X_{\mathcal{L}_i} \subset \mathbb{R}^2$ is a set of points (i.e., x, y coordinates) in $\mathcal{L}_i$ and can be represented, e.g., by the union of polygons, $d_{\mathcal{L}_i} \in \{-1, 1\}$ is the direction of $\mathcal{L}_i$ defined in a consistent manner for all the lanes in $\mathcal{R}$ (i.e., $d_{\mathcal{L}_i} = d_{\mathcal{L}_j}$ iff lanes $\mathcal{L}_i, \mathcal{L}_j$ have the same direction of travel), $v_{\mathcal{L}_i} \in \mathbb{R}$ is the speed limit, $Ent_{\mathcal{L}_i} \subset X_{\mathcal{L}_i}$ is a set of entry points, $Exit_{\mathcal{L}_i} \subset X_{\mathcal{L}_i}$ is a set of exit points, $Stop_{\mathcal{L}_i} \in \{0, 1\}$ indicates whether there is a stop line at the exit of $\mathcal{L}_i$, and $Adj_{\mathcal{L}_i} \subseteq \mathcal{L}$ is a set of lanes that can be entered from every point in $\mathcal{L}_i$ without going through any other lane. Hence, $Adj_{\mathcal{L}_i}$ essentially includes all the lanes that are adjacent to $\mathcal{L}_i$ with no median strip in the middle. For a lane with a stop line at the exit, we assume that the exit points are exactly where the stop line is.

An intersection is defined as a tuple $\mathcal{I}_i = (X_{\mathcal{I}_i}, v_{\mathcal{I}_i}, Exit_{\mathcal{I}_i}, Entry_{\mathcal{I}_i}, Turn_{\mathcal{I}_i})$ where $i \in \{1, \ldots, N_{int}\}$, $X_{\mathcal{I}_i} \subset \mathbb{R}^2$ is a set of points in $\mathcal{I}_i$, $v_{\mathcal{I}_i} \in \mathbb{R}$ is the speed limit, $Exit_{\mathcal{I}_i}$ is a set of exit lanes (i.e., lanes that can be exited through $\mathcal{I}_i$), $Entry_{\mathcal{I}_i}$ is a set of entry lanes (i.e., lanes that can be entered from a lane in $Exit_{\mathcal{I}_i}$ through intersection $\mathcal{I}_i$), and $Turn_{\mathcal{I}_i}$ is a set of tuples that specify the associated pairs of exit and entry lanes. Each tuple in $Turn_{\mathcal{I}_i}$ is of the form $(\mathcal{L}_{j^{ex}}, \mathcal{L}_{j^{en}}, type_j)$ where $\mathcal{L}_{j^{ex}} \in Exit_{\mathcal{I}_i}$, $\mathcal{L}_{j^{en}} \in Entry_{\mathcal{I}_i}$, and $type_j \in \{-1, 0, 1\}$ defines the type of turn (left, straight and right, respectively) from lane $\mathcal{L}_{j^{ex}}$ to $\mathcal{L}_{j^{en}}$.

Next, we define controlled and environment variables.

Controlled Variables: The variables over which the system has control are x, d and v where $x \in \mathbb{R}^2$ is the current position of the vehicle, $d \in \{-1, 1\}$ is the direction defined in a consistent manner with the lane directions, and $v \in \mathbb{R}$ is the speed of the vehicle.

Environment Variables: The environment variables are the positions of the static obstacles and other vehicles. We assume that the environment state can be obtained accurately at run time through the onboard sensors. Let $Obs = Obs_f \cup Obs_b \cup Obs_l \cup Obs_r$ be a set of positions of all the static obstacles where Obs_f and Obs_b include all the static obstacles in our lane, Obs_f include obstacles in front of us whereas Obs_b include obstacles behind our vehicle, and Obs_l and Obs_r include all the static obstacles in the lanes to our left and right, respectively. In addition, we define Veh to be a set of all the other vehicles, identified, e.g., by unique IDs. Let $Loc(Veh)$, $Dir(Veh)$ and $Speed(Veh)$ denote sets of positions, directions and speeds, respectively, of the vehicles in Veh. For any $X \subseteq \mathbb{R}^2$, let $Veh|_X = \{veh \in Veh | Loc(veh) \in X\}$ denote the set of all the vehicles whose positions are within X.

B. Autonomous Vehicle Specification

We use the DUC Technical Evaluation Criteria [9] as the requirement of the vehicle. For the clarity of the presentation, we consider only the case where the vehicle only needs to drive on a road and not in an unstructured region such as parking lot.[1] In this section, we show how each requirement can be expressed in LTL. The overall requirement of the vehicle is simply the logical conjunction of these requirements.

[1]Most of the requirements for those unstructured cases are included in the case we consider in this paper, potentially with different parameters such as vehicle separation. However, many requirements considered in this paper such as staying in a travel lane, passing, stopping at a stop line, intersection precedence, merging and left turn are not included in unstructured cases.

Checkpoints: The vehicle must pass over each checkpoint in the correct lane and in the correct sequence.[2] Assuming that there exists a high-level mission planning component that keeps track of the last checkpoint the vehicle has crossed and issues the next checkpoint to the controller, the corresponding requirement for the traffic planning component is reduced to eventually reaching the next checkpoint. Let ck be the position of the next checkpoint and d_{ck} be the direction of the lane containing the next checkpoint. This requirement can be expressed in LTL as

$$\Diamond(\|x - ck\| \leq \Delta_{ck} \ \wedge \ d = d_{ck}), \tag{1}$$

where Δ_{ck} represents the maximum distance between the vehicle and the checkpoint such that the checkpoint is considered being passed over.

Collisions and vehicle separation: The vehicle maintains a minimum standoff of X_{obs} (e.g., 1 meter) from all obstacles and vehicles in all areas:

$$\Box(dist(x, Obs) \geq X_{obs} \ \wedge \ dist(x, Loc(Veh)) \geq X_{obs}), \tag{2}$$

where for any $x \in \mathbb{R}^2, Y \subset \mathbb{R}^2$, $dist(x, Y) = \inf_{y \in Y} \|x - y\|$ is the closest distance between x and any $y \in Y$.

Speed limits: Vehicle speed conforms to the maximum limit:

$$\begin{array}{l} \bigwedge_{i \in \{1,\ldots,N_l\}} \Box(x \in X_{\mathcal{L}_i} \implies v \leq v_{\mathcal{L}_i}) \wedge \\ \bigwedge_{i \in \{1,\ldots,N_{int}\}} \Box(x \in X_{\mathcal{I}_i} \implies v \leq v_{\mathcal{I}_i}). \end{array} \tag{3}$$

Passing: The vehicle must always come to a full stop and monitor oncoming traffic before crossing a double yellow line. The vehicle maintains a forward vehicle separation of $X_{pass,1}$ when leaving lane to initiate a passing maneuver in a travel area. The vehicle returns to travel lane with vehicle separation between $X_{pass,2}$ and $X_{pass,3}$ when completing a passing maneuver. We define propositions π_p and π_c. Proposition π_p indicates whether there is an obstacle blocking the travel lane such that the vehicle is allowed to perform a passing maneuver. It is initialized to $False$ and evolves according to the following logical dynamical equation

$$\begin{array}{rcl} \pi_p[t+1] & = & \big(\bigvee_i (x[t] \in X_{\mathcal{L}_i} \ \wedge \ d[t] = d_{\mathcal{L}_i}) \wedge \\ & & Obs_f[t] \neq \emptyset \ \wedge \ v[t] = 0 \wedge \\ & & Veh_l[t] = \emptyset\big) \ \vee \ \big(\pi_p[t] \ \wedge \ \neg \pi_c[t]\big), \end{array} \tag{4}$$

where $Veh_l \subseteq Veh$ is a set of all the vehicles in the left lane that are driving in the direction towards our vehicle. Proposition π_c indicates whether the vehicle has completed the passing maneuver. It can be defined as

$$\begin{array}{rcl} \pi_c[t] & = & \big(\bigvee_i (x[t] \in X_{\mathcal{L}_i} \ \wedge \ d[t] = d_{\mathcal{L}_i}) \ \wedge \ Obs_f[t] = \emptyset\big) \\ & & \vee \big(\bigvee_i (x[t] \in X_{\mathcal{L}_i} \ \wedge \ d[t] \neq d_{\mathcal{L}_i}) \wedge \\ & & (Obs_r[t] = \emptyset \ \vee \ (Obs_{r,f}[t] = \emptyset \wedge \\ & & dist(x[t], Obs_{r,b}[t]) \geq X_{pass,3}))\big) \ \vee \ \neg \pi_p[t], \end{array}$$

[2]This statement illustrates ambiguity of requirements written in an informal language. Suppose a given sequence of checkpoint is $c_1, c_2, \ldots$. This statement can be interpreted as the vehicle may not navigate through c_2 before passing over c_1 at all. However, in the DUC, such a behavior is allowed. However, c_2 will not be considered as already being crossed and hence need to be visited again after the vehicle crosses c_1.

where $Obs_{r,f}, Obs_{r,b} \subseteq Obs_r$ are disjoint sets of static obstacles in the right lane that are in front of and behind our vehicle, respectively. Note that π_c, together with π_p, ensures that the vehicle returns to the lane with vehicle separation less than $X_{pass,3}$.

The passing requirement can then be expressed in LTL as

$$\begin{array}{c} \Box\big(\big(\pi_p \ \wedge \ \bigvee_i (x \in X_{\mathcal{L}_i} \ \wedge \ d = d_{\mathcal{L}_i}) \wedge \\ (\pi_p \ \mathcal{U} \ \bigvee_i (x \in X_{\mathcal{L}_i} \ \wedge \ d \neq d_{\mathcal{L}_i}))\big) \implies \\ dist(x, Obs_f) \geq X_{pass,1}\big) \wedge \\ \Box\big(\bigvee_i (x \in X_{\mathcal{L}_i} \ \wedge \ d \neq d_{\mathcal{L}_i}) \implies \\ \big(\bigvee_i (x \in X_{\mathcal{L}_i} \ \wedge \ d \neq d_{\mathcal{L}_i}) \ \mathcal{U} \ (Obs_r = \emptyset \vee \\ (Obs_{r,b} \neq \emptyset \ \wedge \ dist(x, Obs_{r,b}) \geq X_{pass,2}))\big)\big). \end{array} \tag{5}$$

The first and second *always* express the vehicle separation requirements for leaving and returning to lane, respectively.

Stay in lane: The vehicle remains entirely in travel lane at all times except when performing a legal traffic maneuver such as a left turn or maneuvering to avoid an obstacle:

$$\Box\left(\left(\neg \pi_p \ \wedge \ \bigvee_i x \in X_{\mathcal{L}_i}\right) \Longrightarrow \bigvee_i (x \in X_{\mathcal{L}_i} \wedge d = d_{\mathcal{L}_i})\right). \tag{6}$$

Note that this requirement is simplified by the assumption that the vehicle is a point. For vehicles with non-zero length and width, $\mathcal{X}_{\mathcal{L}_i}$ on the right hand side of (6) needs to be changed to a smaller set to account for the size of the vehicle.

Stop line: The vehicle stops within X_{stop} (e.g., 1 meter) of the center of the stop line at intersection. To express this requirement in LTL, we define a proposition π_s that indicates whether the vehicle has stopped at the current stop line. We set $\pi_s[0] = False$ and for $t \geq 0$,

$$\begin{array}{rcl} \pi_s[t+1] & = & \bigvee_i \big(dist(x[t], Exit_{\mathcal{L}_i}) \leq X_{stop} \wedge \\ & & Stop_{\mathcal{L}_i} = 1\big) \ \wedge \ (\pi_s[t] \ \vee \ v[t] = 0). \end{array}$$

The stop line requirement can then be expressed in LTL as

$$\begin{array}{c} \Box\big(\big(\bigvee_i (dist(x, Exit_{\mathcal{L}_i}) \leq X_{stop} \ \wedge \ Stop_{\mathcal{L}_i} = 1)\big) \\ \implies \ (\pi_s \ \vee \ v = 0)\big). \end{array} \tag{7}$$

Intersection precedence and merging: The vehicle respects precedence order at intersections and does not proceed out of turn. In addition, the vehicle always merges into moving traffic when there is a delay of τ_m seconds or more before the arrival of the next traffic-vehicle. Let $L[t] \in \mathcal{L}$ be the lane in which the vehicle is at time t, i.e., $x[t] \in L[t]$. (In the case where the vehicle is in an intersection, we let $L[t]$ be the last lane the vehicle is in.) Also, let $I[t]$ be the intersection with $L[t] \in Exit_{I[t]}$ or $x[t] \in X_{I[t]}$. Note that there can be only one such $I[t]$ due to the assumption that each lane can only exited through one intersection. Finally, let

$$\begin{array}{rcl} X_{st}[t] & = & \bigcup_{\mathcal{L}_j \in Exit_{I[t]}} \{x \in \mathcal{L}_j \mid dist(x, Exit_{\mathcal{L}_j}) \leq X_{stop}\}, \\ X_{ns}[t] & = & X_{\mathcal{I}_t} \cup \\ & & \bigcup_{\substack{\mathcal{L}_j \in Exit_{I[t]} \\ \text{and } Stop_{\mathcal{L}_j} = 0}} \{x \in \mathcal{L}_j \mid dist(x - Exit_{\mathcal{L}_j}) \leq \tau_m v_{\mathcal{L}_j}\}. \end{array} \tag{8}$$

Roughly, $X_{st}[t]$ and $X_{ns}[t]$ define an area around the intersection where we need to monitor oncoming traffic before proceeding through the intersection. Vehicles in $X_{ns}[t]$

have the right of way; thus, we cannot proceed through the intersection while there is a vehicle in $X_{ns}[t]$. In addition, to satisfy the precedence order, we cannot proceed through the intersection while a vehicle that arrives $X_{st}[t]$ before our vehicle has not proceeded. Note that in the computation of $X_{ns}[t]$ above, we assume that the other vehicles also conform to maximum speed limit. This assumption can be relaxed by changing $v_{\mathcal{L}_j}$ to the maximum speed of other vehicles.

Next, we compute the associated sets $Veh_{st}[t]$ and $Veh_{ns}[t]$ of vehicles that should proceed through the intersection before our vehicle. From the merging requirement, we simply set $Veh_{ns}[t] = Veh[t]|_{X_{ns}[t]}$. From the intersection precedence requirement, $Veh_{st}[t]$ can be computed by setting $Veh_{st}[0] = Veh[0]|_{X_{st}[0]}$ and for $t > 0$,

$$Veh_{st}[t] = \begin{cases} Veh[t]|_{X_{st}[t]} & \text{if } Stop_{L[t]} = 1 \wedge \\ & dist(x[t], Exit_{L[t]}) \leq X_{stop} \\ & \wedge\ v[t] = 0\ \wedge\ \neg\pi_s[t] \\ Veh_{st}[t-1] & \text{otherwise} \end{cases}$$

The intersection precedence requirement can then be expressed in LTL as

$$\square\big(\big(\bigvee\nolimits_i (dist(x, Exit_{\mathcal{L}_i}) \leq X_{stop}\ \wedge\ Stop_{\mathcal{L}_i} = 1) \wedge Veh_{ns} \neq \emptyset\ \wedge\ Loc(Veh_{st}) \cap X_{st} \neq \emptyset\big)\implies v = 0\big). \tag{9}$$

Left turn: The vehicle always completes a left turn across a lane carrying oncoming traffic when there is a delay of τ_l or more before the passing of the next oncoming vehicle. For any given time t, we define $X_{left}[t]$ to be an area where we need to monitor the traffic before making the left turn:

$$X_{left}[t] = X_{I[t]} \cup \bigcup_{\substack{\mathcal{L}_j \in Exit_{I[t]} \\ \text{and } Stop_{\mathcal{L}_j} = 0}} \{x \in \mathcal{L}_j \mid dist(x, Exit_{\mathcal{L}_j}) \leq \tau_l v_{\mathcal{L}_j}\}, \tag{10}$$

where $I[t]$ is defined as in (8). In addition, we let $Veh_{left}[t] = Veh[t]|_{X_{left}[t]}$ be the set of all the vehicles that should proceed through the intersection before our vehicle. The left turn requirement can then be expressed in LTL as

$$\square\big(\big(\bigvee\nolimits_{i,j,k} \big(dist(x, Exit_{\mathcal{L}_i}) \leq X_{stop}\ \wedge\ \mathcal{L}_i \in Exit_{\mathcal{I}_j} \wedge ((x \in \mathcal{L}_i\ \vee\ x \in \mathcal{I}_j)\,\mathcal{U}\,(x \in \mathcal{L}_k\ \wedge\ \mathcal{L}_k \in Entry_{\mathcal{I}_j} \wedge (\mathcal{L}_i, \mathcal{L}_k, -1) \in Turn_{\mathcal{I}_j}))\big)\ \wedge\ Veh_{left} \neq \emptyset\big)\implies v = 0\big). \tag{11}$$

IV. EMBEDDED CONTROL SOFTWARE SYNTHESIS

The autonomous driving problem above poses a number of challenges for system designers. First, the system includes a tight coupling between computational and physical elements. In addition, the specification is *reactive* in the sense that the vehicle needs to interact with its environment (e.g., obstacles and other vehicles) and whether it satisfies its specification depends on the behavior of the environments. For example, whether the vehicle exhibits a correct behavior at an intersection depends on the behavior of other vehicles, e.g., which vehicle gets to the intersection first.

A common two-step procedure to the above synthesis problem, as illustrated in Fig. 1, is based on constructing a finite-state abstraction of the underlying physical system (e.g., the motion of the vehicles) and utilizing polynomial-time algorithms for digital design synthesis to synthesize a strategy, represented by a finite state automaton, satisfying high-level specifications. This procedure leads to a hierarchical, two-layer control structure (see Fig. 2) with a discrete planner computing a high-level, discrete plan to be implemented by a low-level, continuous controller. Simulation and bisimulation relations [10] provide a proof that the continuous execution preserves the correctness of the discrete plan. The main limitation of this two-step approach is almost invariably a combinatorial growth of the state space, commonly known as the state explosion problem.

In this section, we provide a brief overview of existing techniques for finite state abstraction of continuous systems and digital design synthesis and a receding-horizon based approach to mitigate the state explosion problem.

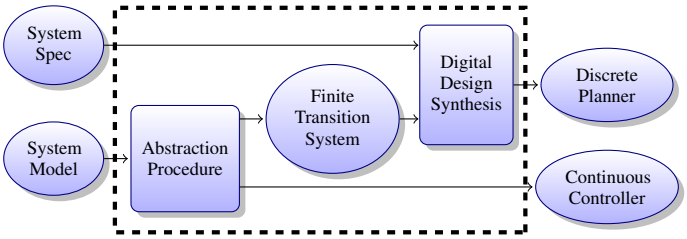

Fig. 1. Two-step procedure for embedded control software synthesis.

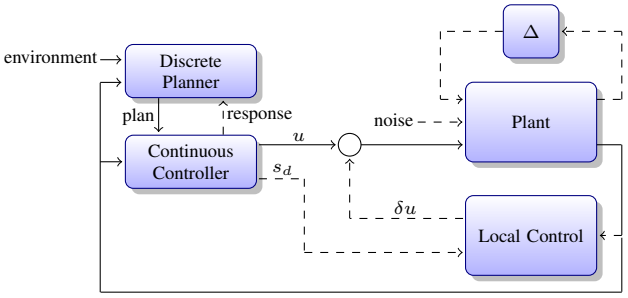

Fig. 2. The two-layer control structure. In addition to the components discussed in this section, Δ, which captures uncertainties in the plant model, may be added to make the model more realistic.

A. Finite State Abstraction of Continuous Systems

There are mainly 2 different ways of abstracting a continuous system into a finite transition systems: fixed abstraction and sampling-based approaches. Fixed abstraction is done offline and is based on partitioning the continuous state space into a finite number of equivalence classes or cells $\nu_1, \ldots, \nu_N$ such that the partition is *proposition preserving* [10]. Roughly speaking, a partition is said to be proposition preserving if for any atomic proposition $p \in \Pi$ and any states v_1 and v_2 that belong to the same cell, v_1 satisfies p if and only if v_2 also satisfies p. Each of $\nu_1, \ldots, \nu_N$ then serves as a state of the finite transition system. We add a transition from ν_i to ν_j where $i, j \in \{1, \ldots, N\}$ if there exists a controller that can bring the system from ν_i to ν_j while always staying in the union of ν_i and ν_j.

Several abstraction methods have been proposed based on a fixed abstraction. For example, a continuous-time, time-invariant model was considered in [11], [12] and [13] for special cases of fully actuated ($\dot{s}(t) = u(t)$), kinematic ($\dot{s}(t) = A(s(t))u(t)$) and piecewise affine dynamics, respectively, while a discrete-time, time-invariant model was

considered in [6] and [14] for special cases of piecewise affine and controllable linear systems, respectively. Discrete-time linear time-invariant state space model with exogenous disturbances was considered in [15], [16]. Reference [17] deals with more general dynamics by relaxing the bisimulation requirement and using the notions of approximate simulation and simulation functions [18].

A sampling-based method, in contrast, incrementally constructs a finite state abstraction of a continuous system online [19]. This construction is based on a random sampling procedure, which can be considered as an extension of the Rapidly-exploring Random Trees (RRT) algorithm. This approach has been used in conjunction with incremental model checking to extract a controller that satisfies a given specification. However, as model checking only provides correctness guarantee with respect to the current environment, it remains an ongoing work to apply the sampling-based approach in conjunction with game theoretic synthesis to ensure system correctness in dynamic environments.

B. Digital Design Synthesis: A Two-Player Game Approach

From Definition 6, for a system to be correct, its specification φ must be satisfied in all of its executions regardless of the behavior of the environment in which it operates. Thus, the environment can be treated as an 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 *Generalized Reactivity(1)* specification, Piterman, et al. shows that checking its realizability and synthesizing the corresponding automaton can be performed in polynomial time [20].

If the specification is realizable, a digital design synthesis tool such as JTLV [20] generates a finite state automaton that represents a set of transitions the system should follow in order to satisfy φ. Otherwise (e.g., in the case of conflicting requirements), it provides the set of initial states starting from which there exists a set of moves of the environment such that the system cannot satisfy φ. Hence, even though the system may be designed manually, it is important to check the realizability of a given specification prior to the design phase as it informs the system designer whether it is possible at all to design a system that meets the given specification.

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 N variables, each can take any value in $\{1, \ldots, M\}$, then there may be as many as M^N nodes in the automaton.

C. Receding Horizon Framework

For systems with a certain structure, the computational complexity of the synthesis problem can be alleviated by solving the problem in a receding horizon fashion, i.e., compute the discrete plan over a shorter horizon, starting from the current state, implement the initial portion of the plan and recompute the plan. This approach essentially reduces the synthesis problem into a set of smaller problems. The size of these smaller problems depends on the horizon length, which should be made as small as possible, subject to the realizability of the resulting short-horizon specifications as too short horizon may render the specifications unrealizable.

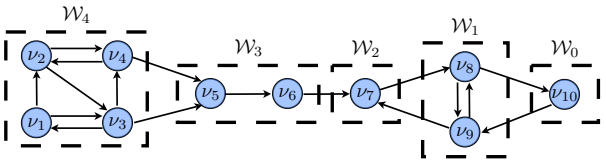

Fig. 3. A graphical description of the receding horizon framework.

In [7], sufficient conditions that ensure that this receding horizon implementation preserves the desired system-level properties are presented. A graphical description of the receding horizon framework is illustrated in Fig. 3. First, we construct a finite state abstraction of the physical system. A partial order relation $\preceq_\varphi$ between the discrete states $\nu_1, \ldots, \nu_N$ needs to be established such that for any destination state ν_k, $\nu_k \prec_\varphi \nu_i$ for all i such that ν_i is not the destination. The disjoint sets $\mathcal{W}_0, \ldots, \mathcal{W}_M$ can then be constructed such that for any discrete states $\nu_i, \nu_j \in \mathcal{W}_k, k \in \{0, \ldots, M\}$, $\nu_i =_\varphi \nu_j$ and $\mathcal{W}_0$ only contains the destination states. Finally, a map $\mathcal{F} : \{\mathcal{W}_0, \ldots, \mathcal{W}_M\} \rightarrow \{\mathcal{W}_0, \ldots, \mathcal{W}_M\}$, which captures the horizon length, and a receding horizon invariant Φ need to be defined.

The receding horizon approach works as follows. Suppose, for example, that the initial state of the system is ν_1. Since $\nu_1 \in \mathcal{W}_4$, we synthesize an automaton satisfying the short-horizon specification where (a) the initial state is assumed to be in $\mathcal{W}_4$ and satisfies Φ, (b) the environment is assumed to satisfy the assumptions stated in the original specification, and (c) the original safety properties are satisfied, Φ holds throughout an execution, and the robot eventually reaches a state in $\mathcal{F}(\mathcal{W}_4)$. The robot then executes this automaton until it reaches a state $\nu_j \prec_\varphi \nu_1$. At this point, the robot computes an automaton for the short-horizon specification associated with the initial state ν_j. This process is then repeated until the robot reaches the destination ν_{10}. We refer the reader to [7] for a detailed discussion on this receding horizon framework.

V. EXAMPLE

We consider a road network of Fig. 4. The checkpoint is located on R_1 as marked by a star in Fig. 4(a). First, we apply a fixed abstraction technique and partition the roads into cells such that there are 2 cells across the width of the road, one completely in the right lane and the other completely in the left lane, as shown in Fig. 4(b). It has been shown in [15]

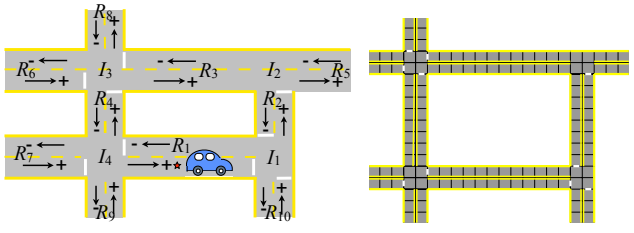

Fig. 4. (a) Road network and (b) its partition.

that this is a valid abstraction of the system, provided that the state of the vehicle evolves based on a fully actuated model $\dot{x}(t) = u(t) + w(t)$ where $u(t), w(t) \in \mathbb{R}^2$ are the control effort and disturbance at time t, respectively, with the constraints $\|u(t)\|_\infty \leq 1$ and $\|w(t)\|_\infty \leq 0.1, \forall t \geq 0$. We then transform the specification in Section III-B such that it is written in terms of cells (as opposed to x, y coordinates as in the original specification).

In order to make the resulting specification realizable, we impose the following assumptions on the behavior of static obstacles and other vehicles: (1) Our vehicle does not collide with obstacles or other vehicles at the initial state; (2) If our vehicle is on the left lane at the initial state, then there is an obstacle on the right lane; (3) The road is not blocked; (4) Obstacles and other vehicles are always detected before our vehicle gets within 1 cell apart from them; (5) Other vehicles do not collide with obstacles; (6) While our vehicle is moving forward, other vehicles do not collide with our vehicle from behind; (7) When our vehicle stops, other vehicles do not collide with our vehicle; (8) Other vehicles respect the stay in lane rule; (9) Other vehicles move infinitely often; (10) Infinitely often, $Veh_{ns} = \emptyset$; and (11) Infinitely often, there is no oncoming traffic on the left lane.

Next, we apply the receding horizon framework. To this end, a software toolbox TuLiP [21] is employed. First, TuLiP verifies that a planning horizon of length 2 cells is sufficiently long and all the resulting short horizon specifications with this horizon are realizable. The partial order relation as well as the map $\mathcal{F}$ and the receding horizon invariant can be constructed as in [7]. The finite state automata generated by TuLiP contain between 571 and 2434 states and take between 0.7 and 6.5 seconds to compute on a MacBook Pro with a 2.8 GHz Intel Core 2 Duo processor. Snapshots of a simulation run is shown in Fig. 5 for different scenarios. The video of this run can be accessed at `http://dl.dropbox.com/u/29005314/sim.zip`.

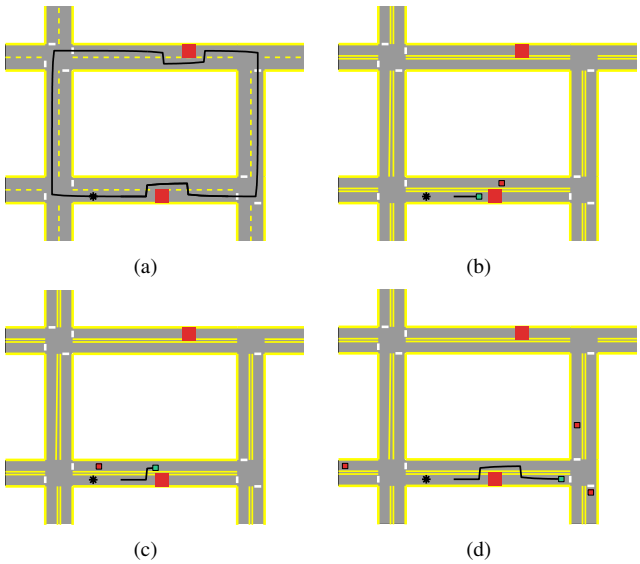

Fig. 5. Simulation results. (a) Trajectory of the vehicle. (b) Vehicle stops and wait for the oncoming traffic before performing a passing maneuver. (c) Vehicle performs a passing maneuver after encountering an obstacle that blocks the travel lane. (d) Vehicle stops at a stop line and waits for its turn.

REFERENCES

[1] S. M. Loos, A. Platzer, and L. Nistor, "Adaptive cruise control: Hybrid, distributed, and now formally verified," in *FM* (M. Butler and W. Schulte, eds.), vol. 6664 of *LNCS*, pp. 42–56, Springer, 2011.

[2] T. Wongpiromsarn, S. Mitra, R. M. Murray, and A. Lamperski, "Periodically controlled hybrid systems: Verifying a controller for an autonomous vehicle," in *Hybrid Systems: Computation and Control* (R. Majumdar and P. Tabuada, eds.), vol. 5469 of *LNCS*, pp. 396–410, Springer, 2009.

[3] E. A. Emerson, "Temporal and modal logic," in *Handbook of theoretical computer science (vol. B): formal models and semantics*, pp. 995–1072, Cambridge, MA, USA: MIT Press, 1990.

[4] H. Kress-Gazit and G. J. Pappas, "Automatically synthesizing a planning and control subsystem for the DARPA Urban Challenge," in *IEEE International Conference on Automation Science and Engineering*, pp. 766–771, 2008.

[5] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas, "Courteous cars: Decentralized multi-agent traffic coordination," *Robotics and Automation Magazine*, vol. 15, no. 1, pp. 30–38, 2008.

[6] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proc. of IEEE Conference on Decision and Control*, 2009.

[7] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *International Conference on Hybrid Systems: Computation and Control*, 2010.

[8] A. Pnueli, "Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends," *Current Trends in Concurrency. Overviews and Tutorials*, pp. 510–584, 1986.

[9] "Urban Challenge technical evaluation criteria." Defense Advanced Research Projects Agency, 2006.

[10] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, 2000.

[11] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 3116–3121, April 2007.

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

[13] 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.

[14] 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.

[15] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Automatic synthesis of robust embedded control software," in *AAAI Spring Symposium on Embedded Reasoning: Intelligence in Embedded Systems*, pp. 104–111, 2010.

[16] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*. submitted.

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

[18] 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.

[19] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. of IEEE Conference on Decision and Control*, 2009.

[20] 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://jtlv.sourceforge.net/.

[21] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "TuLiP: A software toolbox for receding horizon temporal logic planning," in *International Conference on Hybrid Systems: Computation and Control*, 2011. Software available at `http://sourceforge.net/projects/tulip-control/`.