Correct, Reactive Robot Control from Abstraction and Temporal Logic Specifications

Hadas Kress-Gazit, *Member, IEEE,* Tichakorn Wongpiromsarn, and Ufuk Topcu, *Member, IEEE*

***Abstract*—We describe recent advances in formal synthesis of robot controllers from temporal logic specifications. In particular, we consider reactive specifications where the robot continuously gathers information about its environment and decides its action at run time based on this information. The automatically generated controller is provably correct with respect to a given specification for all the valid environment behaviors. We discuss the main limitation of such controller synthesis – the state explosion problem – and two approaches that mitigate this problem. Computational tools that implement these approaches are also described. An autonomous vehicle navigating an urban-like environment is used as an illustrative example throughout the paper.**

***Index Terms*—Temporal logic, high-level control, receding horizon**

I. INTRODUCTION

Robots these days feature a tight interplay between computational and physical components. Take the vehicles in the recent DARPA Urban Challenge (DUC) [1] as an example. The competing vehicles were supposed to navigate, in a fully autonomous manner, through a partially known urban-like environment populated with static and dynamic obstacles and perform different tasks such as on-road and off-road driving, parking and visiting certain areas while obeying traffic rules. They also needed to negotiate intersections, handle changes in the environment or operating conditions and re-plan in response to such changes. During the execution of these high-level tasks, the vehicles had to obey the low-level physics and actuation limitations that constrain their maneuverability. Hence, the high-level logic that governs the behavior of the vehicles needs to be properly integrated with the low-level controller that regulates the physical hardware.

Systems with tight coupling between computations and physics are typically complex, yet a lot of them are designed and implemented in an ad-hoc manner. Even though correct operation of the individual components is established through extensive tests or formal verification, the complexity of the overall system usually renders testing impractical and exceeds the capabilities of formal verification tools. For example, a mismatch in the abstraction of the physical system used at different levels of the planning stack of Alice (Figure 1) [2], Team Caltech's entry in the DUC, had never been discovered in thousands of hours of our simulations and over three hundred miles of field testing [3]. Once the problem was uncovered, it was difficult to modify the underlying ad-hoc design. In fact, this problem triggered the series of unacceptable behavior that disqualified Alice from the 2007 Challenge. Therefore, we maintain that successful deployment of autonomous vehicles and robots in general will require advances in formal approaches both for verification and correct-by-construction synthesis of embedded controllers.

Fig. 1: Alice, Team Caltech's entry in the 2007 DUC.

In this paper, we review recent advances in formal synthesis of embedded robot controllers. Specifically, we consider the following problem. Given a model for the robot and its specification expressed in a formal language, here Linear Temporal Logic (LTL), synthesize a control protocol that, by construction, ensures that the continuous robot behavior satisfies the given specification for all valid environment behaviors. We address *reactive* specifications in which the robot behavior is different depending on the information it gathers through its sensors at run time. For example, if a robot encounters a blocked road the controller reacts to it by changing the robot's trajectory so that it still reaches a certain checkpoint.

A common two-step procedure to the above synthesis problem is based on constructing a finite-state abstraction of the underlying physical system (e.g., the motion of the vehicles in the autonomous driving example) and synthesizing a strategy [4], represented by a finite state automaton, that satisfies high-level specifications. This procedure leads to a hierarchical, two-layer control structure with a discrete planner computing a high-level, discrete plan which is implemented by a low-level, continuous controller. Simulations and bisimulation relations [5] provide a proof that the continuous execution preserves the correctness of the discrete plan. The main limitation of the approaches for synthesizing reactive control protocols is almost invariably the resulting computational

H. Kress-Gazit is with the Sibely School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, 14850 USA e-mail: hadaskg@cornell.edu.

T. Wongpiromsarn is with the Singapore-MIT Alliance for Research and Technology, Singapore 117543 email: nok@smart.mit.edu.

U. Topcu is with with the Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA email: utopcu@cds.caltech.edu.

complexity. In this paper we describe two approaches that mitigate this problem. One relies on coarse abstractions that are constructed based on appropriate low-level controllers and aims to synthesize complete controllers, thus providing global guarantees for task completion. The other approach assumes a fine-grain abstraction more suitable for complex dynamics and employs a receding horizon framework where a controller only plans out an execution for a short step ahead and re-computes the plan as the robot moves, thus allowing for complex dynamics at the expense of completeness.

Construction of controllers that ensure that the system satisfies certain temporal logic specifications has recently attracted considerable attention. References [6]–[8] exploit the availability of powerful model checking tools to synthesize such control protocols. These approaches typically generate open-loop strategies, i.e., they cannot satisfy reactive behaviors that depend on the current state of the environment or handle initial condition uncertainties. In [9] the authors transform LTL specifications into a mixed-integer linear problem to solve multi-robot missions. In [10] the authors generate control policies that can deal with actuation error while probabilistically satisfying LTL specifications. There, the specification is not reactive but the behavior is reactive with respect to disturbances. In this paper, on the other hand, we address reactivity at the specification and control level; however, we assume robust controllers that can deal with disturbances at the low level.

The paper is structured as follows. We first provide useful definitions and descriptions of the formalisms and algorithms. We discuss, through the use of an illustrative example of an autonomous vehicle driving in an urban-like setting, the specification, control synthesis and execution of the robot behavior. We then describe two approaches that clarify the tradeoffs inherent in dealing with the computational complexity and describe tools that are available. We conclude with future directions and challenges.

II. Background

Addressing the problem of generating continuous control for the motion and action of a robot operating in the physical world, the work described in this paper first abstracts the problem into a discrete problem, then obtains a provably correct discrete solution and finally continuously implements that solution. To facilitate the discussion we first define the main formalisms that are used:

A. Automata

An automaton[1] $\mathcal{A}$ is a tuple $\mathcal{A} = (Q, Q_0, \Sigma, \Gamma, \delta, \gamma)$ where

- Q is a set of states,
- $Q_o \in Q$ is the set of initial states,
- Σ is an input alphabet (set of symbols used to label transitions),
- Γ is an output alphabet (set of symbols used to label states),
- $\delta : Q \times 2^{\Sigma} \to 2^{Q}$ is the transition relation, i.e., $\delta(q, X) = Q'$ where $Q' \subseteq Q$ and $X \subseteq 2^{\Sigma}$ is a set of input symbols, defines the set of all possible next states given a current state q and an input X, and
- $\gamma : Q \to 2^{\Gamma}$ is the labeling function that assigns to each state a set of output symbols $\gamma(q) = Y, Y \subseteq 2^{\Gamma}$.

In the following, such an automaton will represent the discrete solution of the robot control problem where the input alphabet corresponds to the robot's sensor information and the output alphabet to the robot's motion and action.

A run r of the automaton $\mathcal{A}$ under an input sequence $x(0), x(1) \ldots, x(i) \in 2^{\Sigma}$ is a sequence of states $q(0), q(1), \ldots$ that satisfy (a) $q(0) \in Q_0$ and (b) $q(i+1) \in \delta(q(i), x(i))$ for all $i \geq 0$.

We define the behavior of a run $r = q(0), q(1), \ldots$ under an input sequence $x(0), x(1) \ldots$, denoted as $behav(r)$, as the sequence $(x(0), y(0)), (x(1), y(1)), \ldots$ such that $x(i) \in 2^{\Sigma}, y(i) = \gamma(q(i))$ for all $q(i) \in r,\ i \geq 0$.

B. Linear Temporal Logic

The work described in this paper utilizes Linear Temporal Logic (LTL) as the underlying formalism that is used to specify high-level, reactive robot behavior. Loosely speaking, LTL allows one to reason about the change over time of the truth values of atomic propositions. The syntax of LTL is recursively defined as follows, where π is an atomic proposition in the set AP which also include the $True$ and $False$ propositions:

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

Given negation ($\neg$) and disjunction ($\vee$), we can define conjunction ($\wedge$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$). The temporal operators "next" ($\bigcirc$) and "until" ($\mathcal{U}$) can be used to derive additional temporal operators such as "eventually" $\Diamond\varphi = True\,\mathcal{U}\,\varphi$ and "always" $\Box\varphi = \neg\Diamond\neg\varphi$.

The semantics of an LTL formula φ is defined on an infinite sequence σ of truth assignments to the atomic propositions $\pi \in AP$. In this paper, we are interested in LTL formulas that are defined over the set of infinite behaviors generated by a robot's control automaton.

Informally, the formula $\bigcirc\varphi$ expresses that φ is true in the next position in the sequence, i.e., the infinite behavior starting from the next state in the automaton satisfies φ. The formula $\varphi_1 \,\mathcal{U}\, \varphi_2$ expresses the property that φ_1 is true until φ_2 becomes true. The sequence σ satisfies formula $\Box\varphi$ if φ is true in every position of the sequence (i.e., in every state), and satisfies the formula $\Diamond\varphi$ if φ is true at some position of the sequence (in some reachable state). Sequence σ satisfies the formula $\Box\Diamond\varphi$ if φ is true infinitely often. For a formal definition of the semantics we refer the reader to [12].

C. LTL Synthesis

Synthesizing an LTL formula φ refers to the process of automatically generating an automaton $\mathcal{A}$ such that, for every infinite run of the automaton, the behavior of the run satisfies the formula[2], i.e., $behav(r) \models \varphi\ \forall r$. It has been shown that

[1]We slightly abuse the term automaton here as we do not define the usual set of accepting states. A more accurate term would be "Kripke Structure [11] with labeled transitions".

[2]Note that in the formal methods community, the synthesized automata do not usually have labeled transitions (e.g. [4]), i.e., Σ is empty, $\delta : Q \to 2^{Q}$ and Γ contains the environmental (input) propositions.

the complexity of synthesizing an arbitrary LTL formula is double exponential in the size of the formula [13] but if the specification is restricted to a subset of LTL, the complexity becomes polynomial in the state space [4].

In the following we show how this restricted subset of LTL can be used to capture many robotics tasks and how the synthesis algorithm of [4] is used in a global and receding horizon manner. Informally, the synthesis algorithm transforms the specifications into a game between the environment and the robot. If the robot can win no matter what the environment does, an automaton is generated. If, on the other hand, the environment can prevent the robot from achieving its goals or the specification is unsatisfiable (i.e., there is some logical contradiction), the algorithm will return that the specification is unrealizable. For more details regarding the algorithm the reader is referred to [4].

Tools: There are several tools that implement LTL synthesis algorithms. For example, Lily [14], [15] can be used to synthesize arbitrary LTL formulas but due to the complexity it can handle only small specifications with few variables, and RATSY [16] is a synthesis and analysis tool. In the work described in this paper we have been using JTLV [17], [18], which is a tool for implementing formal methods algorithms and implementations of the synthesis algorithm of [4].

III. Specification and Control Synthesis of Robotic Tasks

In the previous sections we defined the formalism (LTL) and the formal methods used to create a discrete solution (automaton synthesis). We now discuss how these formal methods are used within the context of correct-by-construction, high-level robot control. We will use the following example to illustrate the techniques and tradeoffs.

Example 1: Consider an autonomous driving problem in an urban-like environment, similar to the DUC. Important desired properties of the vehicle typically include obstacle avoidance, staying in the right lane, obeying traffic rules and reaching different destinations. These properties can be easily expressed in LTL [19], [20]. In the following sections we describe two approaches to generating control for such an autonomous vehicle that abstract the problem at different levels and provide different guarantees.

A. From Specifications to Discrete Solution

At the core of the described approach is the need to abstract continuous, infinite-state systems into equivalent (in the simulation sense) atomic propositions and finite state models. Several abstraction methods have been proposed based on a fixed abstraction of motion. For example, a continuous-time, time-invariant model was considered in [21], [22], [23] and [24] for special cases of fully actuated ($\dot{s}(t) = u(t)$), kinematic ($\dot{s}(t) = A(s(t))u(t)$) and piecewise-affine (PWA) dynamics, respectively. A discrete-time, time-invariant model was considered in [25] and [26] for special cases of PWA and controllable linear systems, respectively. Reference [27] deals with more general dynamics by relaxing the bisimulation requirement and using the notion of approximate simulation [28].

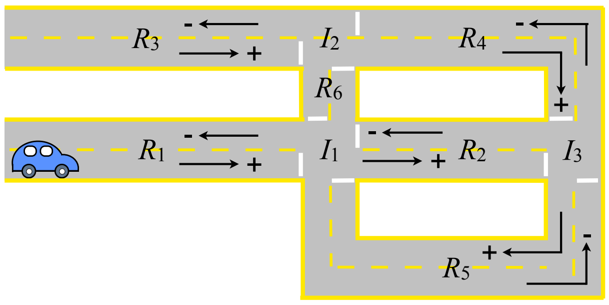
In order to capture a robotic task using LTL we define a set of atomic propositions that contains three types of propositions:

- **Sensor propositions** s_i**:** These propositions abstract the perception system of the robot and provide information about the environment.
- **Location propositions** r_i**:** These propositions are used to abstract the position of the robot in the workspace (whether or not the robot is inside a defined region).
- **Action propositions** a_i**:** These propositions abstract actions the robot can perform; for example, $signalRight$ that represents the on/off action of activating the right turn signal.

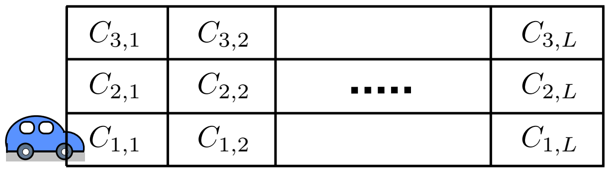
Note that the sensor propositions represent the behavior of the environment and the robot has no control over their value.

We represent the set of environment (sensor) propositions as $\mathcal{X} = \{s_1, \ldots s_m\}$. The location and action propositions represent the robot behavior and are set by the robot controller. We represent the set of robot propositions as $\mathcal{Y} = \{r_1, \ldots r_n, a_1, \ldots, a_k\}$.

Once the problem is abstracted using a set of atomic propositions, the robot task can be written as a conjunction of LTL formulas of a given structure [4], [22]. These formulas capture: (i) initial conditions of the environment and the robot, (ii) the motion capabilities of the robot by encoding the possible transitions between regions of the workspace, (iii) any assumptions, about the behavior of the sensor propositions, i.e., about the behavior of the environment, and (iv) the restrictions, conditions and goals for the robot's behavior.



(a) A coarse partition of a network of roads and intersections



(b) A fine-grain partition of a two-lane road into cells

Fig. 2: Different levels of abstraction for the motion of an autonomous vehicle

Example 2: Consider the autonomous driving problem described in Example 1. We consider different levels of abstraction. For example, in a coarse abstraction (Figure 2(a)), a location proposition R_1 can be used to indicate whether the robot is currently traveling on road R_1. A sensor proposition

$RoadBlocked$ can represent the binary output (blocked/not blocked) of a combined vision and lidar system that reasons about the state of the road. A sensor proposition $redLight$ can represent the perception system alerting the robot that there is a red traffic light at the intersection. An action proposition $stop$ can cause the robot to employ the breaks and stop its motion. Examples for LTL formulas representing the desired behavior are

- $\Box(R_2 \implies (\bigcirc R_2 \vee \bigcirc I_1 \vee \bigcirc I_3))$ expresses that from R_2 the robot can either stay in R_2 or move to I_1 or I_3.
- $\Box(\bigcirc redLight \implies \bigcirc stop)$ expresses that the robot should stop at a red light.
- $\Box\Diamond R_4$ expresses that the robot should eventually reach R_4.

In a finer abstraction, suppose R_1 has two lanes. It can be partitioned as shown in Figure 2(b) where cells $C_{1,1}, \dots, C_{1,L}$ are completely in the right lane, $C_{2,1}, \dots, C_{2,L}$ cover the center of the road and $C_{3,1}, \dots, C_{3,L}$ are completely in the left lane. Here, for each $i \in \{1, 2, 3\}, j \in \{1, \dots, L\}$, $C_{i,j}$ is a location proposition whereas $o_{i,j}$, which represents the existence of an obstacle in cell (i, j), is a sensor proposition. Examples of LTL formulas are

- $\bigwedge_{i,j} \Box\big(o_{i,j} \implies \neg C_{i,j}\big)$ is the obstacle avoidance requirement.
- $\bigwedge_j \Box\big(\neg(o_{1,j} \vee o_{1,j-1} \vee o_{1,j+1}) \implies (\neg C_{2,j} \wedge \neg C_{3,j})\big)$ where $o_{1,0}, o_{1,L+1} \equiv False$ is the staying in lane requirement.

Since the robot does not control the environment propositions, the synthesis algorithm needs to take into account all the possible values of $redLight$ (in the coarse abstraction case), $o_{i,j}$ (in the finer abstraction case) and the resulting behavior of the system needs to satisfy the specification (LTL formulas as listed above) for all these values.

B. From Discrete Solution to Continuous Execution

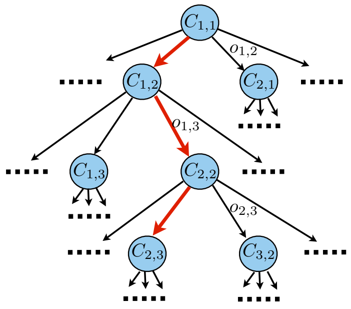
The synthesized automaton $\mathcal{A}$ that satisfies the LTL formulas (synthesized based on [4]) is implemented as a hybrid controller for the robot as described in Algorithm 1. Given $\mathcal{A}$, an LTL formula $\varphi_{initial}$ describing the initial condition of the propositions in the current execution and an initial pose for the robot p_0, the initial state $q_0 \in Q$ is determined and the robot actions are enabled/disabled. Then, at each iteration, the value of the environmental propositions $\mathcal{X}$ determine what the next state $NxtState$ and the next region $NxtReg \in \gamma(NxtState)$ should be. The controller then invokes a continuous controller that drives the robot towards the next region. If the robot enters the next region, the current automaton state changes and the appropriate actions are enabled/disabled. If the robot is neither in the current region nor in the next region, which could only happen if the environment violated its assumptions, the execution is stopped with an error. Note that unlike traditional discrete automaton execution in which transitions between states are instantaneous, here the transitions usually correspond to the robot moving between regions and therefore take time.

Algorithm 1 Continuous Execution of the discrete solution

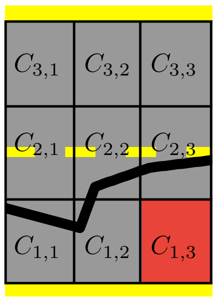
```
procedure EXECUTE($\mathcal{A}$, $\varphi_{initial}$, $p_0$)
    $q_0 \leftarrow \{q \mid \gamma(q) \models \varphi_{initial}\}$
    $CurrState \leftarrow q_0$
    $CurrReg \leftarrow r_i \in \gamma(q_0)$
    $CurrOutput \leftarrow \{a_1, a_2, \cdots\} \in \gamma(q_0)$
    $ActivateDeactivateOutputs(CurrOutput, \mathcal{Y})$
    $p \leftarrow p_0$
    while 1 do                                          ▷ Infinite execution
        $InputVal \leftarrow$ Value of sensor propositions $\mathcal{X}$
        $NxtState \leftarrow$
                $GetNxtState(\mathcal{A}, CurrState, InputVal)$
        $NxtReg \leftarrow r_i \in \gamma(NxtState)$
        $p \leftarrow ApplyController(CurrReg, NxtReg, p)$
        if $p \in$ Region corresponding to $NxtReg$ then
                                                        ▷ Automaton transition
            $CurrState \leftarrow NxtState$
            $CurrReg \leftarrow NxtReg$
            $CurrOutput \leftarrow \{a_1, a_2, \cdots\} \in \gamma(CurrState)$
            $ActivateDeactivateOutputs(CurrOutput, \mathcal{Y})$
        else if $p \in$ Region corresponding to $CurrReg$ then
                                                        ▷ No transition
            Continue
        else
            ERROR - No legal automaton state
        end if
    end while
end procedure
```

Simulations/bisimulations [5] provide a proof that the continuous execution preserves the correctness of the discrete plan. Informally, for the discrete solution to be implementable, we require a set of continuous controllers that are able to correctly implement any discrete transition in the controller automaton. For example, if there is a transition from a state in which $C_{1,1}$ is true (the robot is in cell $C_{1,1}$) to a state in which $C_{1,2}$ is true, there must be a controller able to drive the robot from any point within cell $C_{1,1}$ across the boundary to cell $C_{1,2}$ without going through any other cell along the way. Clearly, the abstraction of the robot dynamics has to be created such that the set of controllers exists and furthermore, the robustness of such controllers will determine the granularity of the abstraction. We refer to the continuous controllers as 'bisimilar' controllers since they correspond to a bisimulation between the discrete and continuous models; every discrete transition in the abstraction can be implemented in the continuous world (ensures correctness) and every continuous transition between regions in the workspace is encoded in the discrete abstraction (ensures completeness with respect to the partition).

Example 3: Going back to Example 2, Figure 3(a) depicts part of the synthesized automaton generated based on the specifications for the fine abstraction. Figure 3(b) displays the resulting trajectory of the car. While driving along the right lane, the car senses an obstacle. The execution of the automaton (the red transitions) follows the transition that is labeled with $o_{1,3}$ and that causes the car to move to the next lane.



(a) Snippet of the synthesized automaton. The red transitions correspond to the specific run.



(b) Example execution, in which an obstacle is detected in cell $C_{1,3}$ ($o_{1,3}$ becomes true).

Fig. 3: Automaton and execution of the behavior of an autonomous vehicle from Example 2.

C. Challenges

The two main challenges for this type of formal approach are (a) finding a suitable **abstraction** that allows the problem to be encoded and solved at the discrete level while being correctly implementable in the continuous domain and (b) dealing with the **state explosion** problem. These two challenges are clearly related since the coarser the abstraction, the more compact the state space and vice versa.

In the following sections we illustrate this tradeoff in the context of Example 1. First we discuss an approach that relies on a coarse abstraction and a set of robust continuous feedback controllers thus allowing the full controller to be synthesized, i.e., every possible environmental behavior and the correct corresponding robot control is encoded in it. Furthermore, we show an example in which the specification can be naturally decomposed and synthesized into several automata such that the sequential execution satisfies the original specification, thus mitigating the state-space explosion problem. Then, we describe an approach that relies on a fine-grain abstraction that captures more detailed specification and that employs a receding horizon framework to alleviate the associated computational complexity of LTL synthesis.

IV. Coarse Abstractions: Synthesizing the Full Controller

One approach to deal with the challenges of synthesis is to create a coarse abstraction for the motion of the robot where the workspace is partitioned into different regions of interest with high-level semantics such as roads, intersections and parking spaces in the case of autonomous vehicles, or rooms and hallways in the case of a robot operating indoors.

As described above, the model (abstraction) of the robot motion in the workspace is captured in the LTL specifications and is embedded in the synthesized discrete automaton, restricting the motion of the robot. In order to generate the continuous control for the robot, there must be a set of controllers that are able to continuously implement every discrete transition of the discrete automaton. In general, given an abstraction, finding such a set of controllers for robots with arbitrary complex dynamics is an unsolved problem. However, there have been several control schemes developed for different dynamics such as fully actuated point robots [29]–[31], nonholonomic [32], [33] and groups of robots [34]. In the context of an autonomous vehicle, the work of Conner et. al. [35] can be used to generate a pallet of local control policies for nonholonomic, convex-bodied vehicles that correspond to driving in a lane, turning at an intersection and performing a parallel parking maneuver. The policies can be automatically instantiated over a known map [23], [36].

Revisiting Example 1, we now show how the high-level behavior of the robot can be captured if one assumes the existence of robust feedback controllers that are able to drive a nonholonomic vehicle along roads and through intersections (turn left/right) as discussed above. Specifically, here we focus on the high-level behavior of the vehicle: correctly traversing 4-way stop intersections, exhibiting e-stop behavior, avoiding obstacles and road blocks and going through a prescribed set of checkpoints [19].

Framing the problem in the context of the DUC, for each mission the robot was given two text files describing the challenge. The Route Network Definition File (RNDF) contains the description of the course, road segments, lanes, waypoints and locations of stop signs and checkpoints. The second file, given to the team five minutes before the start of the mission, is the Mission Data File (MDF) containing the sequence of checkpoints to be traversed together with speed limits for the different road segments. In addition to this, the robot was expected to follow the California traffic laws regarding right of way, etc. In [19] we have shown how the DUC mission can be encoded in structured English that was then transformed to LTL formulas and synthesized into a correct-by-construction controller. There, in order to deal with the state explosion problem, we decomposed the problem into four sets of disjoint specifications and then generated four automata that, when composed, generated the correct behavior. Figure 4 shows the structure of the composed automaton. Note that the road behavior layer was synthesized once and could be reused as long as the traffic laws did not change. The driving control, which included the motion abstraction as encoded in the RNDF and the checkpoints encoded in the MDF, was generated for different such files.

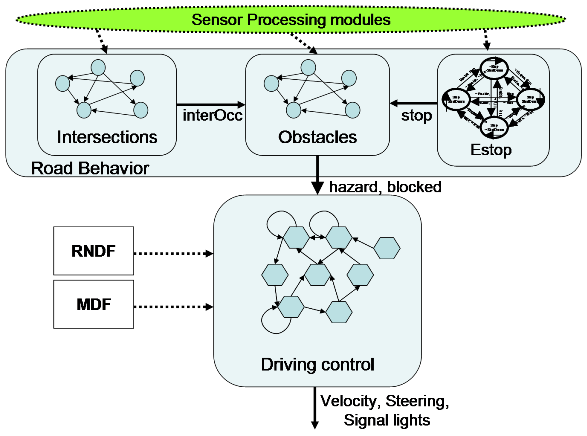
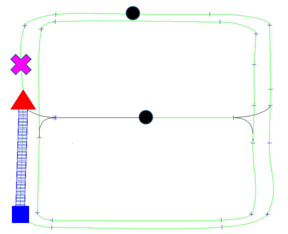
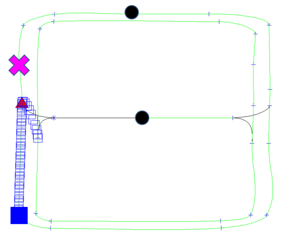


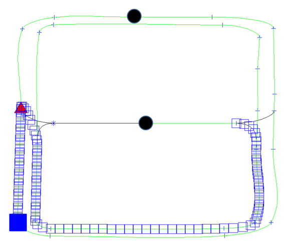
Fig. 4: Structure of discrete automaton for the DUC example

Figures 5 and 6 depict simulations of the synthesized system. Since the focus was on the high-level behavior (i.e., robust feedback controllers were assumed), the dynamics of the vehicle were abstracted by motion on a line representing the road. Figure 5 depicts the behavior of a robot on one of the National Qualifying Event (NQE) maps, area A, where the robot had to go through two checkpoints (denoted by black circles) as many times as possible. In this simulation, the robot detects an obstacle and after waiting for a predefined time decides that the road is blocked and moves on to an alternate route. Figure 6 depicts the behavior of the robot in area C of the NQE. There the robot exhibits appropriate behavior at a 4-way stop first allowing the green car to go through and then moving through the intersection.

We note that the controllers used in the simulation were automatically generated from the same set of LTL specifications and the appropriate RNDF and MDF files. The behavior is different in part due to the information the robot receives at run time such as obstacles and cars at intersections. Furthermore, any desired change in the high-level behavior is simple to implement (change some of the sentences) and the correctness guarantees are preserved.


(a) Stopping due to obstacle (robot stopped is indicated by red triangle)


(b) Timer timed out. Obstacle is determined to be a blocked road


(c) Robot takes a different route

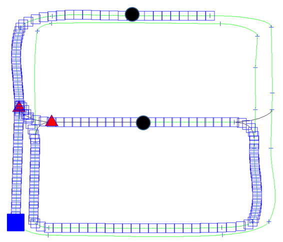
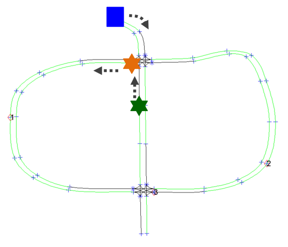
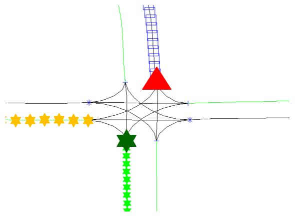

(d) Block cleared

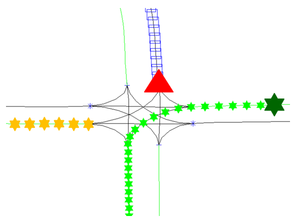
Fig. 5: Encountering a temporarily blocked road

Tool: We have built Linear Temporal Logic MissiOn Planner (LTLMoP) [37], an open-source Python-based toolbox for the design, synthesis and implementation of high-level robot control from structured English specifications. Some of the toolbox features are as follows.

- Modular design to accommodate different research advancements (such as in the control, language interface, automata synthesis, etc.).
- A graphical user interface for drawing the regions of interest of a workspace (RegionEditor),
- A structured English grammar for writing specifications that supports non-projective locative prepositions such as 'between' and 'within x' [38],
- Connection to Player/Stage [39] for simulating and controlling physical robots.


(a) Initial locations of cars. The robot is depicted using a blue square


(b) Both the robot and the green car stop (robot stopped is indicated by red triangle)


(c) The green car moves

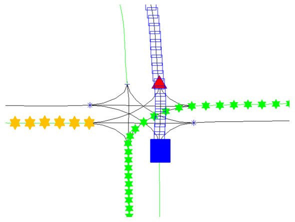
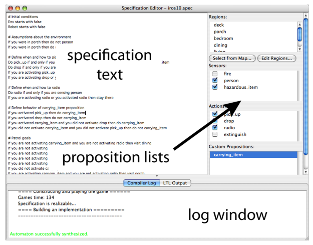
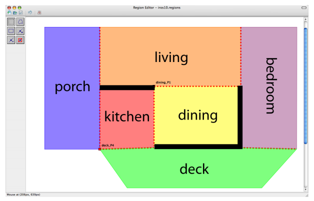

(d) The robot moves

Fig. 6: 4-way stop behavior

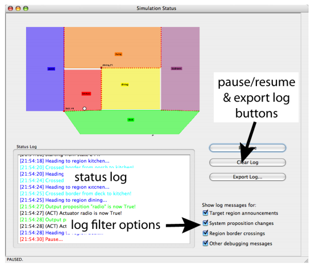
Figure 7 depicts several screen shots of LTLMoP and a photo of a robot being controlled by it. Videos showing LTLMoP in action can be seen at http://www.youtube.com/user/ASLCornell.



(a) SpecEditor, the main window in which the propositions are displayed, the specification written and a log of the automaton synthesis displayed. All features are accessed through this window.



(b) RegionEditor, a graphical user interface for drawing the regions of interest and boundary of the environment.



(c) Simulation window. The robot is the white circle moving around the map. The log window provides the state of the actions and other information.

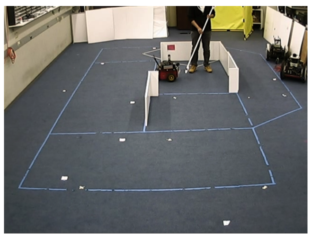

(d) a Pioneer robot being controlled from LTLMoP. The robot reacts correctly to the information it gathers about the environment at runtime.

Fig. 7: Screen shots of LTLMoP (images from [37])

V. Fine abstraction: Receding horizon framework

To illustrate the state explosion problem, we revisit the autonomous driving problem of Example 2. Consider the case where the road is partitioned as in Figure 2(b). Suppose the car starts in cell $C_{1,1}$ and the destination is the union of $C_{1,L}$, $C_{2,L}$ and $C_{3,L}$. In this problem, there are $3L2^{3L}$ possible states of the system. The computational complexity of the algorithm presented in [4] is $O(|\mathcal{V}|^3)$ where $|\mathcal{V}|$ is the size of the state space, which, in this case, is exponential in L. This type of computational complexity limits the application of LTL synthesis to relatively small abstractions. Furthermore, when the complicated dynamics of an autonomous ground vehicle needs to be incorporated, the road may need to get further discretized [25], resulting in an even larger state space.

In many applications, however, it is not necessary for the robot to plan for the whole execution, taking into account all the possible behaviors of the environment, since a state that is very far from the current state of the robot typically does not affect the near future plan. In the context of the autonomous driving problem of Example 2, under certain conditions, it may be sufficient for the robot to plan out an execution for only a short segment ahead and implement it in a receding horizon fashion, i.e., re-compute the plan as the robot moves, starting from the currently observed state (rather than from all the possible initial conditions).

In [20], sufficient conditions that ensure that this receding horizon implementation preserves the desired system-level properties are presented. This framework reduces the computational complexity of the synthesis problem by essentially breaking the original problem into a set of smaller problems of shorter horizon. The size of these smaller problems depends on the horizon length. For example, consider the autonomous driving problem of Example 2 where the robot starts in cell $C_{1,1}$ and the destination is the union of $C_{1,L}$, $C_{2,L}$ and $C_{3,L}$. Suppose the horizon length is l (i.e., the robot plans for l cells ahead). Then, the state space for each short-horizon problem contains at most $3l2^{3l}$ states (whereas the size of the original problem is $3L2^{3L}$). Hence, the horizon length should be made as small as possible, subject to the realizability of the resulting short-horizon specifications. A horizon that is too short typically renders the specifications unrealizable. For the previously mentioned autonomous driving problem, it was shown in [20] that all the short-horizon specifications are realizable with $l = 2$. Hence, the size of the state space for each short-horizon problem is at most 384 regardless of the length L of the road while for $L = 100$, the size of the state space of the original problem is on the order of 10^{92} and it increases exponentially with L.[3]

The correctness of this receding horizon framework relies on a partial order relation among the discrete states. The notion of a receding horizon invariant, a proposition that needs to remain true throughout the execution, was introduced in order to ensure that a provably correct plan exists when the robot reaches the end of the current horizon and needs to compute a new plan (i.e., the robot does not end up in a "bad" state where it cannot proceed without violating some desired properties).

A graphical description of the receding horizon framework is illustrated in Figure 8 for a special case where there is only one destination. First, a partial order relation $\preceq_\varphi$ between the discrete states 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 $\mathcal{W}_0, \dots, \mathcal{W}_M$ can then be constructed such that for any discrete states $\nu_i, \nu_j \in \mathcal{W}_k, k \in \{0, \dots, M\}$, $\nu_i =_\varphi \nu_j$ and $\mathcal{W}_0$ only contains the destination states. Finally, a map $\mathcal{F} : \{\mathcal{W}_0, \dots, \mathcal{W}_M\} \to \{\mathcal{W}_0, \dots, \mathcal{W}_M\}$, 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 $\nu_1 \in \mathcal{W}_4$, the robot synthesizes 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 ν_{10}, which is the destination. We refer the reader to [20] for a detailed discussion on this receding horizon framework, including an extension to the case where there are multiple destinations that may be reached in an arbitrary order.

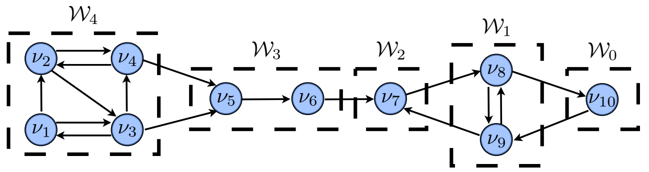


Fig. 8: A graphical description of the receding horizon framework for a special case where there is only one destination. $\nu_1, \dots, \nu_{10}$ are the discrete states.

Computation of the horizon length, partial order relation and receding horizon invariant requires insights for each problem domain. Automatic construction of these elements is subject to current research. Reference [20] describes automatic construction of certain elements, given other elements, e.g., automatic computation of the horizon length and partial order relation, given a receding horizon invariant, and automatic computation of the receding horizon invariant, given a horizon length and partial order relation.

The receding horizon approach is not complete. Even if the original specification is realizable, there may not exist a combination of horizon length, partial order relation and receding horizon invariant satisfying the sufficient conditions presented in [20]. Nevertheless, its successful applications to autonomous driving problems have been illustrated in [20], [25], [40]. Examples of the results are provided in Figure 9.

Tool: TuLiP [3], a Python-based toolbox for embedded control software synthesis, implements the previously described receding horizon framework. The key features of TuLiP include embedded control software synthesis and receding horizon planning. The synthesis feature relies on (1) generating a proposition preserving partition of the continuous state space,

[3]Note that the specification considered in [20] also includes intersection rules, which are not included in the straight road scenario considered here. Hence, the size of the state space reported in [20] is slightly larger than the size of the state space of the problem considered here. However, the same horizon length applies for both scenarios.

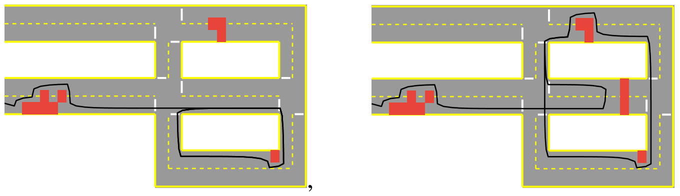


Fig. 9: Simulation results with (left) no road blockage, (right) a road blockage on the middle road. The corresponding movies can be downloaded from http://www.cds.caltech.edu/tulip.

(2) continuous state space discretization based on the evolution of the continuous state [40], and (3) digital design synthesis. JTLV [4] is used as the underlying synthesis routine.

Currently, TuLiP handles the case where the continuous state of the robot evolves according to discrete-time linear time-invariant dynamics: for $t \in \{0, 1, 2, \ldots\}$,

$$s[t+1] = As[t] + Bu[t] + Ed[t], u[t] \in \mathcal{U}, d[t] \in \mathcal{D}, s[0] \in \mathcal{S}$$

where $\mathcal{S} \in \mathbb{R}^n$ is the continuous state space, $\mathcal{U} \in \mathbb{R}^m$ and $\mathcal{D} \in \mathbb{R}^p$ are the sets of admissible control inputs and exogenous disturbances, $s[t], u[t], d[t]$ are the continuous state, the control signal and the exogenous disturbance, respectively, at time t. $\mathcal{U}, \mathcal{D}, \mathcal{S}$ are assumed to be bounded polytopes.

Successful applications of TuLiP include autonomous driving [25], vehicle management systems in avionics [41] and multi-target tracking [42]. Other simpler examples are included in the current release of the toolbox.

VI. Conclusions and Future Challenges

In this paper we presented an overview of how temporal logic synthesis, coupled with abstractions and continuous bisimilar controllers can be used to generate high-level, reactive robot control. We illustrated the ideas using the DARPA Urban Challenge mission and we presented two approaches to dealing with the inherent state explosion problem.

There are several directions worth pursuing in the context of logic synthesis for robot control. These include relaxing different assumptions such as perfect sensing and actuation, addressing the question of optimality, dealing with uncertainty, increasing the size of systems and specifications that can be synthesized, and finding appropriate abstractions based on the robot dynamics, workspace and mission.

Acknowledgements

This work was partially supported by ARO MURI SUBTLE W911NF-07-1-0216, AFOSR and the Boeing Corporation.

References

[1] "DARPA Urban Challenge," http://www.darpa.mil/grandchallenge/index.asp, 2007.

[2] J. W. Burdick, N. DuToit, A. Howard, C. Looman, J. Ma, R. M. Murray, and T. Wongpiromsarn, "Sensing, navigation and reasoning technologies for the DARPA Urban Challenge," DARPA Urban Challenge Final Report, Tech. Rep., 2007.

[3] 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, submitted, software available at http://www.cds.caltech.edu/tulip.

[4] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) Designs," in *VMCAI*, Charleston, SC, January 2006, pp. 364–380.

[5] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, pp. 971–984, 2000.

[6] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion: State of the art and grand challenges," *Robotics and Automation Magazine*, vol. 14, no. 1, pp. 61–70, 2007.

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

[8] *Sampling-Based Motion Planning with Temporal Goals.* Anchorage, Alaska: IEEE, May 3, 2010 2010.

[9] S. Karaman and E. Frazzoli, "Complex mission optimization for multiple-uavs using linear temporal logic," in *American Control Conference*, Seattle, Washington, 2008.

[10] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta, "Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees," in *ICRA*, 2010, pp. 3227–3232.

[11] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. Cambridge, Massachusetts: MIT Press, 1999.

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

[13] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM Press, 1989, pp. 179–190.

[14] B. Jobstmann and R. Bloem, "Linear logic synthesizer (lily)," http://www.ist.tugraz.at/staff/jobstmann/lily/ 2006.

[15] ——, "Optimizations for ltl synthesis," in *FMCAD '06: Proceedings of the Formal Methods in Computer Aided Design*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 117–124.

[16] R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Könighofer, M. Roveri, V. Schuppan, and R. Seeber, "Ratsy - a new requirements analysis tool with synthesis," in *CAV*, 2010, pp. 425–429.

[17] Y. Sa'ar, "Java temporal logic verifier (jtlv)," http://jtlv.ysaar.net/ 2008.

[18] A. Pnueli, Y. Saar, and L. Zuck, "Jtlv: A framework for developing verification algorithms," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, T. Touili, B. Cook, and P. Jackson, Eds. Springer Berlin / Heidelberg, 2010, vol. 6174, pp. 171–174.

[19] H. Kress-Gazit and G. J. Pappas, "Automatically synthesizing a planning and control subsystem for the darpa urban challenge," in *IEEE Conference on Automation Science and Engineering*, Washington D.C., USA, 2008.

[20] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proc. of the 13th International Conference on Hybrid Systems: Computation and Control*, 2010.

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

[22] ——, "Temporal logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[23] D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas, "Valet parking without a valet," in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, San Diego, CA, October 2007, pp. 572–577.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of Local Potential Functions for Global Robot Control and Navigation," in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, Las Vegas, NV, October 2003, pp. 3546 – 3551.

[30] C. Belta and L. Habets, “Constructing decidable hybrid systems with velocity bounds,” in *IEEE Conference on Decision and Control*, Bahamas, 2004.

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

[32] S. R. Lindemann, I. I. Hussein, and S. M. LaValle, “Realtime feedback control for nonholonomic mobile robots with obstacles,” in *IEEE Conference on Decision and Control*, San Diego, CA, 2006.

[33] D. C. Conner, H. Choset, and A. Rizzi, “Towards provable navigation and control of nonholonomically constrained convex-bodied systems,” in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA ’06)*, May 2006.

[34] N. Ayanian and V. Kumar, “Decentralized feedback controllers for multi-agent teams in environments with obstacles,” in *IEEE International Conference on Robotics and Automation*, Pasadena, CA, May 2008.

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

[36] 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.

[37] C. Finucane, G. Jing, and H. Kress-Gazit, “Ltlmop: Experimenting with language, temporal logic and robot control,” in *IEEE/RSJ Int’l. Conf. on Intelligent Robots and Systems*, Taipei, Taiwan, October 2010, pp. 1988 – 1993.

[38] H. Kress-Gazit and G. J. Pappas, “Automatic synthesis of robot controllers for tasks with locative prepositions,” in *IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, 2010, pp. 3215–3220.

[39] B. Gerkey, R. T. Vaughan, and A. Howard, “The player/stage project: Tools for multi-robot and distributed sensor systems,” in *11th International Conference on Advanced Robotics ICAR*, Coimbra, Portugal, June 2003, pp. 317–323.

[40] 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*, 2010, pp. 104–111.

[41] ——, “Formal synthesis of embedded control software: Application to vehicle management systems,” in *AIAA Infotech@Aerospace*, 2011, submitted.

[42] N. Ozay, U. Topcu, T. Wongpiromsarn, and R. M. Murray, “Distributed synthesis of control protocols for smart camera networks,,” in *International Conference on Cyber-Physical Systems*, 2011, submitted.