Dynamics-Driven Adaptive Abstraction for Reactive High-Level Mission and Motion Planning

Jonathan A. DeCastro[1], Vasumathi Raman[2] and Hadas Kress-Gazit[1]

***Abstract*— We present a new framework for reactive synthesis that considers the dynamics of the robot when synthesizing correct-by-construction controllers for nonlinear systems. Many high-level synthesis approaches employ *discrete abstractions* to reason about the dynamics of the continuous system in a simplified manner. Often, these abstractions are expensive to compute. We circumvent the need to have detailed abstractions for nonlinear systems by proposing a framework for adapting abstractions based on partial solutions to the low-level controller synthesis problem. The contribution of this paper is a reactive synthesis algorithm that makes use of our adaptation procedure to update the high-level strategy each time the non-deterministic discrete abstraction is modified. We combine this with a verified low-level controller synthesis scheme capable of automatically synthesizing controllers for a wide class of nonlinear systems. This novel synthesis framework is demonstrated on a dynamical robot executing an autonomous inspection task.**

I. Introduction

The synthesis problem takes as input a discrete abstraction of a robot, a map of its workspace, and a formal mission specification, and returns (if possible) a high-level controller that achieves the specified behavior. In reactive synthesis [11], [22], the specification and abstraction include a description of the uncontrolled environment in which the robot operates. In order to be implemented on real robotic platforms, these approaches rely on the computation of *atomic controllers*, which are feedback control policies verified to implement the various high-level actions of the synthesized controller in finite time. The task of constructing these low-level controllers given a particular robotic platform presents a major challenge to synthesis. Recent tools for such multi-layered synthesis have included provably-correct online motion planning [2], [14], automatically-constructed navigation functions [9], and offline synthesis of verified controllers for a wide class of nonlinear systems [7]. Each of these methods preserve a hierarchical planning structure: whether or not the strategy can be implemented rests heavily on the dynamics of the physical system, as well as the availability of an accurate abstraction.

The following example emphasizes the importance of accurately modeling the robot's dynamics during the high-level synthesis step. Consider the map in Figure 1, where a robot moving with constant speed and limited turning radius is assigned the following task: *Start in* $r1$. *Visit* $r2$. *If a person is seen when in* $r1$, *remain in* $r1$. *Assume that infinitely often no person will be seen. Assume that if not in* $r1$, *no person will be seen.* A high-level controller satisfying the task (Figure 1a) mandates that robot stay in $r2$ once it is reached; however, this cannot be implemented at the low level given the robot's physical limitations for movement in $r2$. Nonetheless, there exists a discrete controller (Figure 1b) that both satisfies the specification and can be implemented at the low level given the robot's dynamics. The atomic controllers that implement this solution steer the robot from $r1$ to either $r2$ or $r3$ if no person is seen.

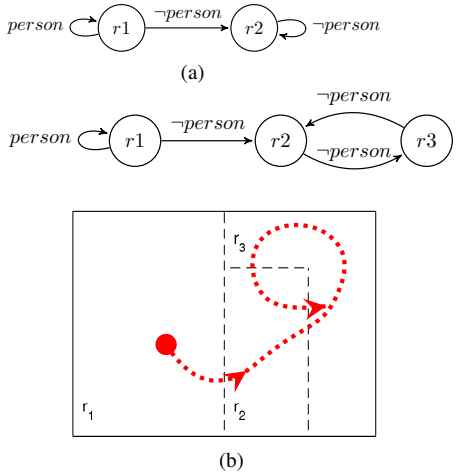


Fig. 1: (a) High-level controller that cannot be implemented at the low-level due to inaccurate abstraction of the robot dynamics. States are labeled with regions and edges are labeled with environment inputs. (b) Realizable high-level controller and continuous trajectory.

In contrast to strictly hierarchical planning methods, we introduce a framework that tightly couples the synthesis of high-level controllers with the automatic computation of atomic controllers. As opposed to approaches that use a fixed discrete abstraction for temporal logic synthesis, we continually adapt the abstraction based on partial solutions to the atomic controller synthesis problem, in combination with formal analysis at the high level. We first introduce a framework for encoding classes of abstractions that assume arbitrary motion durations. Given that a high-level controller has been synthesized, a set of constraints is imposed on the atomic controllers. Our scheme then attempts to implement the strategy by synthesizing these atomic controllers, revising the discrete abstraction if any actions are deemed unachievable during the process. We use the intermediate results of the high-level synthesis to further guide modifications to the

*J.A. DeCastro and H. Kress-Gazit are supported in part by NSF Expeditions in Computer Augmented Program Engineering (ExCAPE). V. Raman is supported in part by TerraSwarm.

[1]Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA {jad455,hadaskg}@cornell.edu;

[2]Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena CA 91125, USA vasu@caltech.edu.

abstraction. If successful, the algorithm outputs both a high-level controller and a family of atomic controllers that satisfy the specification. The focus of this paper is robot locomotion, and extending the approach to actions besides motion is the subject of future work.

Numerous works have dealt with provably-correct controller synthesis while accounting for complex system dynamics. For example, in [13], [12] two methods are introduced for synthesizing reactive switching policies that are versatile enough to handle a wide class of abstractions, while in [10] a method is introduced for synthesizing high-level controllers using a custom-designed set of low-level controllers for a vehicle with Ackermann steering. In the non-reactive setting, recent work has harnessed reachability computations guided by a synthesized control strategy, for the purpose of refining that strategy. Notably, [3] introduce a method where a tree-based motion planner is used to explore a coarsely-decomposed workspace. Any exploration results are fed back to the high-level planner to update the mission plan based on certain "feasibility estimates" that, in part, account for the dynamics of the robot. The authors of [14] extend this work in the development of an iterative planning strategy for nonlinear systems in which uncertain elements in the environment may be discovered at runtime. The approach taken in [21] leverages the results of high-level synthesis to specify a set of constrained reachability problems to be solved by an optimizer, and uses this information to concretize an abstract high-level plan.

Drawing inspiration from previous approaches to counterexample-guided abstraction refinement [6], we adapt our discrete abstractions on the fly, eliminating the need for the potentially costly procedure of computing up front a catch-all discrete abstraction for a given dynamical system. Others (e.g. [15], [1]) have used specifications to guide refinement of the discrete abstraction, relying on an incremental re-partitioning of the workspace. This partitioning step could expand the number of problem variables in an unbounded fashion, increasing complexity of the high-level synthesis. Instead, we adopt an abstraction methodology based on a fixed workspace partitioning, which does not result in this explosion of added variables.

Rather than computing a concrete trajectory to instantiate an abstract one as in [3], [21], we adopt the problem setting of [7] in which we compute atomic controllers that can be invoked at runtime, with each one verified over a finite domain of the state space. Hence, our approach differs from that of [2], [3], in that we are able to provide formal guarantees that allow for immediate reactivity to the sensed environment. Our approach is also less conservative than [7] with respect to specifications that can be realized on a given robot platform due to the fact that our algorithm adapts the abstraction when the low-level synthesis step fails. That is, if a specification is realizable and can be implemented on a given robot model, it is more likely to find such controllers using the approach presented here.

II. Problem Formulation

A. Dynamical System

Consider the function $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ defining the (deterministic) system

$$\dot{\xi}(t) = f(\xi(t), u(t)), \tag{1}$$

where $\xi(t)$ is the continuous state of the robot and $u(t)$ the command input of the robot at time $t \in \mathbb{R}_{\geq 0}$. Denote by ξ_T the trajectory defined over the finite time interval $t \in [0, T)$ starting from an initial state $\xi_T(0)$.

B. Linear Temporal Logic

The syntax of linear temporal logic (LTL) formulas is defined over a set AP of atomic (Boolean) propositions by the recursive grammar:

$$\varphi ::= \pi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

where $\pi \in AP$ and $\wedge$, $\neg$, $\bigcirc$, and $\mathcal{U}$ are the operators "conjunction", "negation", "next", and "until", respectively. We derive "disjunction" $\vee$, "implication" $\Rightarrow$, "equivalence" $\Leftrightarrow$, "always" $\Box$, and "eventually" $\Diamond$ from these operators. LTL formulas are evaluated over infinite sequences $\sigma = \sigma_0\sigma_1\sigma_2\ldots$ of truth assignments to the propositions in AP. For example, σ satisfies "always" φ, denoted $\sigma \models \Box\varphi$, if and only if φ is true in every σ_i. The reader is referred to [5] for further details on the semantics of LTL.

We define a set of *environment* propositions $\mathcal{X}$ describing sensed events that the robot must react to, and a set of *system* propositions $\mathcal{Y}$ describing the actions of the robot; $AP = \mathcal{X} \cup \mathcal{Y}$. For $S \subseteq AP$, we also define $\bigcirc S = \{\bigcirc \pi_i\}_{\pi_i \in S}$. Finally, we denote by 2^S the set of truth valuations to the Boolean propositions in S (each subset of S corresponds to a truth valuation in which that set of variables is `True` and everything else is `False`).

Definition 1 (*Robot Mission Specification*). In this work, a *robot mission specification* is an LTL formula of the form:

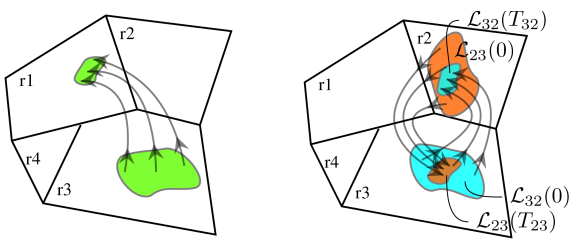
$$\varphi := \varphi^e \implies \varphi^s,$$

where φ^e and φ^s are defined over AP and $\bigcirc AP$, and are further decomposed into formulas for initial conditions, safety conditions to be satisfied always, and liveness conditions (goals) to be satisfied infinitely often[1].

Definition 2 (*Controller Strategy*). Define a *controller* as a finite-state machine $\mathcal{A} = (S, S_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma^{\mathcal{X}}, \gamma^{\mathcal{Y}})$, where

- S is the set of controller states;
- $S_0 \subseteq S$ is the set of initial controller states;
- $\mathcal{X}$ and $\mathcal{Y}$ are environment and system propositions;
- $\delta : S \times 2^{\mathcal{X}} \to S$ is a state transition relation;
- $\gamma^{\mathcal{X}} : S \to 2^{\mathcal{X}}$ is a labelling function mapping controller states to the set of environment propositions that are `True` for incoming transitions to that state, and;
- $\gamma^{\mathcal{Y}} : S \to 2^{\mathcal{Y}}$ is a labelling function mapping controller states to the set of system propositions that are `True` in that state.

[1]More precisely, robot mission specifications are in the the generalized reactivity fragment of rank 1 (GR(1)).



(a) (b)

Fig. 2: (a) A transition of the discrete abstraction. (b) Two sequentially-composable transitions.

Given an infinite sequence of truth valuations to environment propositions, $X^\omega = x_0x_1x_2\ldots \in (2^{\mathcal{X}})^\omega$, the execution of $\mathcal{A}$ starting from s_0 is given by $s_0s_1s_2\ldots$ where $s_i = \delta(s_{i-1}x_{i-1})$ for all $i \geq 1$. The sequence of labels (truth valuations to AP) thus generated is $\sigma^{\mathcal{A}}_{X^\omega} = (\gamma^{\mathcal{X}}(s_0), \gamma^{\mathcal{Y}}(s_0))(\gamma^{\mathcal{X}}(s_1), \gamma^{\mathcal{Y}}(s_1))\ldots$. An LTL formula φ is *realizable* if there exists a controller $\mathcal{A}$ such that, for every $X^\omega \in (2^{\mathcal{X}})^\omega$, $\sigma^{\mathcal{A}}_{X^\omega} \models \varphi$. If there does not exist such an $\mathcal{A}$, φ is *unrealizable*. Synthesizing $\mathcal{A}$ (if it exists) from a robot mission specification follows directly from the solution to a two-player game, as described in [4].

C. Discrete Abstractions

Given a bounded configuration space $W \subset \mathbb{R}^n$, let $\mathcal{R} = \{r_1 \ldots r_p\}$ represent a set of regions, not necessarily disjoint, covering W, where $r_i \subseteq W$. We adopt the motion encoding of [18] by introducing the set of *completion* propositions $\mathcal{X}_c \subseteq \mathcal{X}$ that are `True` whenever a transition between regions has completed, or a request to remain within a region is already fulfilled. Let $\pi_i \in \mathcal{X}_c$ denote a proposition that is `True` when the robot is in $r_i \in \mathcal{R}$, and let $\pi_{ai} \in \mathcal{Y}$ denote a proposition that is `True` when the robot is *activating* an atomic controller to move to $r_i \in \mathcal{R}$. We assume that all π_{ai} are mutually exclusive – the robot can only activate motion towards one region at a time. For example, Figure 2a shows a case where, if the robot is in r_3 (π_3 is `True`) and is activating π_{a1}, it will eventually arrive in r_1.

Definition 3 (***Controlled Discrete Abstractions***)**.** We define a *controlled discrete abstraction* S_r as the tuple $(\mathcal{X}_c, \mathcal{Y}, \delta_r, \Pi_{inv})$, where:

- $\mathcal{X}_c$ and $\mathcal{Y}$ are atomic propositions defined as above;
- $\delta_r : \mathcal{X}_c \times \mathcal{Y} \to \mathcal{X}_c$ is a transition relation defining a target region $\pi_j = \delta_r(\pi_i, \pi_{aj})$ given a region $\pi_i \in \mathcal{X}_c$ and action $\pi_{aj} \in \mathcal{Y}$.
- $\Pi_{inv} : \mathcal{X}_c \times \mathcal{Y} \to 2^{\mathcal{X}_c}$ is a function mapping a region $\pi_i \in \mathcal{X}_c$ and action $\pi_{aj} \in \mathcal{Y}$ to a set of possible regions that may be visited during the transition from π_i to π_j under π_{aj}.

In Figure 2a, for example, $\delta_r(\pi_3, \pi_{a1}) = \pi_1$ and $\Pi_{inv}(\pi_3, \pi_{a1}) = \{\pi_3, \pi_1, \pi_2\}$ because there are a total of three regions that could be visited upon invoking a transition from r_3 under the action π_{a1}.

D. Atomic Controllers

Let $I(i,j)$ denote an index set of invariant regions for transitioning from π_i to π_j under the action π_{aj}, such that $\pi_i, \pi_j \in \Pi_{inv}(\pi_i, \pi_{aj}) = \{\pi_k \mid k \in I(i,j)\}$. Also, assume that each region indexed by $I(i,j)$ is connected to at least one other region indexed by $I(i,j)$. Given any two regions with associated completion propositions π_i, π_j and an action π_{aj}, we define an *atomic controller* $\kappa_{i,j} : \mathbb{R}^n \to \mathbb{R}^m$ as a function mapping continuous states to control commands $u(t) = \kappa_{ij}(\xi_{T_{ij}}(t))$ such that, given the initial condition $\xi_{T_{ij}}(0) \in r_i$ there exists a final state $\xi_{T_{ij}}(T_{ij}) \in r_j$ such that $\xi_{T_{ij}}(t) \in \bigcup_{k \in I(i,j)} r_k$ for all $t \in [0, T_{ij})$. Intuitively, κ_{ij} denotes a controller that produces a trajectory steering the system from r_i to r_j without leaving the set defined by the regions enumerated in the set $\Pi_{inv}(\pi_i, \pi_{aj})$. Returning to Figure 2a, κ_{31} produces the set of trajectories shown, driving the system from r_3 to r_1 while remaining in $\Pi_{inv}(\pi_3, \pi_{a1}) = \{\pi_3, \pi_1, \pi_2\}$.

We denote the *reach set* of κ_{ij} by $\mathcal{L}_{ij}(t) \subset \bigcup_{k \in I(i,j)} r_k$, for $t \in [0, T_{ij})$. $\mathcal{L}_{ij}$ is defined as the time-indexed set such that, for all initial states $\xi_{T_{ij}}(0) \in \mathcal{L}_{ij}(0)$, for all $t \in [0, T_{ij})$ the continuous state of the robot satisfies $\xi_{T_{ij}}(t) \in \mathcal{L}_{ij}(t)$ under the controller command $u(t) = \kappa_{ij}(\xi_{T_{ij}}(t))$ for all $t \in [0, T_{ij})$. We denote a collection of atomic controllers by $\mathcal{C}$, where $\mathcal{C} = \{\kappa_{ij}\}_{i,j}$.

Definition 4 (***Low-Level Controller Implementations***)**.** Given a controller strategy $\mathcal{A}$ and $s \in S$, let $\pi_i = \gamma^{\mathcal{X}}(s) \cap \mathcal{X}_c$ be the region the robot is in for state s. Define

$$I^i_{out} = \{k \mid s' = \delta(s, \gamma^{\mathcal{X}}(s')), \pi_k = \gamma^{\mathcal{X}}(s') \cap \mathcal{X}_c\}.$$

I^i_{out} is the index set of all possible successor regions to π_i in $\mathcal{A}$. Also, let σ_{pre} denote a finite prefix of an execution σ, and σ_{suff} denote an infinite suffix of σ. For $j \in I^i_{out}$, a family of controllers $\mathcal{C}$ is an *implementation* of $\mathcal{A}$ if, for all executions $\sigma = (\sigma_{pre}(\gamma^{\mathcal{X}}(s), \pi_i)(\gamma^{\mathcal{X}}(s'), \pi_j)\sigma_{suff})$ of $\mathcal{A}$, the following two properties are satisfied (Definitions 1 and 2 in [7]):

1) **Sequential composition.** $\mathcal{L}_{ij}(T_{ij}) \subseteq \bigcap_{k \in I^j_{out}} \{\mathcal{L}_{jk}(t)\}_{t \in [0, T_{jk})}$. That is, κ_{jk} is sequentially composable if each of the successor transitions is reachable from the target set of its predecessor $\mathcal{L}_{ij}(T_{ij})$.
2) **Reactive composition.** For each $k \in I^j_{out}$ and for all $t \in [0, T_{jk})$, if there is a trajectory $\xi_{T_{jk}}$ such that $\xi_{T_{jk}}(t) \in \mathcal{L}_{jk}(t) \cap r_j$, then $\xi_{T_{jk}}(t) \in \bigcap_{k \in I^j_{out}} \{\mathcal{L}_{jk}(t)\}_{t \in [0, T_{jk})}$. Intuitively, everywhere along the trajectory in r_j, the robot may 'opt-out' of moving to region k and instead enable an action to a region $I^j_{out} \backslash \{k\}$.

Figure 2b illustrates sequential composability of κ_{23} with respect to κ_{32} because $\mathcal{L}_{23}(T_{23}) \subset \mathcal{L}_{32}(0)$. We say that $\mathcal{A}$ is *implementable* under f if there exists an infinite continuous trajectory controlled by a sequence $\mathcal{C}$ for all executions σ of $\mathcal{A}$; otherwise, it is *unimplementable*. Note that these definitions imply that the infinite continuous trajectories are non-blocking; that is, the robot will always have access to a valid controller allowing for all of the executions in $\mathcal{A}$.

E. Problem Statement

The problem addressed in this paper is the following:

Problem 1. Given a continuous system f and a formula φ that is realizable with respect to a discrete abstraction S_r, synthesize a high-level controller $\mathcal{A}$ and an implementing library of low-level atomic controllers $\mathcal{C}$. If no low-level controllers can be computed, find a new S_r (if one exists) for which both $\mathcal{A}$ and $\mathcal{C}$ can be synthesized.

III. Encoding Abstractions as LTL Formulas

In this section, we show how to transform non-deterministic controlled discrete abstractions in Definition 3 into a set of temporal logic formulas, representing the behavior of continuous systems where the completion of motion commands take arbitrarily long to complete. To accomplish this, we adopt an approach similar to that described in [18].

We transform S_r in Definition 3 into a formula ψ^r over $\mathcal{Y}$ and $\mathcal{X}_c$ that encodes the allowed robot commands given the current region. In this setting, we require two sets of formulas: one describing the transitions allowed by S_r; the other describing completion of a motion. These formulas are defined as follows:

$$\psi_t^r = \bigwedge_{\pi_i \in \mathcal{X}_c} \Box \left(\bigcirc \pi_i \implies \bigvee_{\substack{\pi_{aj} \in \mathcal{Y}: \\ \delta_r(\pi_i, \pi_{aj}) \neq \emptyset}} \bigcirc \pi_{aj} \right), \tag{2}$$

$$\psi_{tc}^r = \bigwedge_{\substack{\pi_i \in \mathcal{X}_c \\ \pi_{aj} \in \mathcal{Y}}} \Box \left(\pi_i \wedge \pi_{aj} \implies \bigvee_{\pi_j \in \Pi_{inv}(\pi_i, \pi_{aj})} \bigcirc \pi_j \right) \tag{3}$$

Here ψ_t^r describes which region propositions can be activated in the next time step ($\bigcirc \pi_{aj}$) given the next completion variables ($\mathcal{X}\pi_i$). On the other hand, ψ_{tc}^r describes the allowed transitions in terms of the completion variables that can become true in the next time step ($\bigcirc \pi_j$) given the current completion variables (π_i) and the motion controllers that are currently active (π_{aj}).

Finally, we define

$$\psi_g^r = \Box \diamondsuit \bigvee_{\pi_j \in \mathcal{X}_c, \pi_{aj} \in \mathcal{Y}} (\pi_{aj} \wedge \bigcirc(\pi_j \vee \neg \pi_{aj})). \tag{4}$$

This is a fairness assumption on the environment, which enforces that every action eventually completes as long as the system doesn't change its mind.

For example, consider the transition shown in Figure 2a. Taking π_{a1}, π_{a2} as activation propositions and $\pi_1, \ldots, \pi_4$ as completion propositions we have the following:

$$\psi_t^r = \Box(\pi_3 \implies (\bigcirc \pi_{a1} \vee \bigcirc \pi_{a2})),$$

$$\begin{aligned} \psi_{tc}^r = \Box(\pi_3 \wedge \pi_{a1} &\implies (\bigcirc \pi_3 \vee \bigcirc \pi_1 \vee \bigcirc \pi_2)) \wedge \\ \Box(\pi_3 \wedge \pi_{a2} &\implies (\bigcirc \pi_3 \vee \bigcirc \pi_2)) \wedge \\ \Box(\pi_2 \wedge \pi_{a1} &\implies (\bigcirc \pi_2 \vee \bigcirc \pi_1)), \end{aligned}$$

$$\psi_g^r = \Box \diamondsuit ((\pi_{a1} \wedge \bigcirc(\pi_1 \vee \neg \pi_{a1})) \vee (\pi_{a2} \wedge \bigcirc(\pi_2 \vee \neg \pi_{a2}))).$$

Due to the non-determinism in the abstraction, the formulas ψ_{tc}^r and ψ_g^r act as additional assumptions on the environment, while ψ_t^r is a specification for the system. The environment in this case is allowed to choose any successor state in the abstraction as long as the assumptions are fulfilled. We obtain the final specification

$$\varphi' = (\varphi^e \wedge \psi_{tc}^r \wedge \psi_g^r) \implies \varphi^s \wedge \psi_t^r.$$

This specification is a slight generalization of GR(1), since the system liveness condition admits the $\bigcirc$ operator; the synthesis algorithm in [17] can be used to synthesize an automaton for φ'.

Note that our formulation of the abstraction treats a similar class of systems as in [13], [12], but our contribution has some significant differences. First, the liveness condition requires, for each action, that infinitely often, a region transition will be completed. In contrast, [13] offers a means for explicitly characterizing the behaviors of trajectories (controlled or uncontrolled) and as such only require a liveness property that states that the trajectories must always leave a region or set of regions. This difference emphasizes the point that our abstractions encode additional information on the convergence of the *controlled* system trajectories. The second difference is that we allow abstractions to encode controllers that reach a region *via* other regions rather than just neighboring regions, as long as other specified constraints on the system behavior hold. In this regard, our approach can be thought of as a generalization of [13], [12].

IV. Synthesis via Adaptation of the Discrete Abstraction

We now outline an approach for solving Problem 1 using the encoding described above. Our approach generalizes to any type of reachability-based motion planning technique, including those using barrier certificates [16], differential games [8], and LQR trees [19]. The only requirement is the ability to compute an invariant set of continuous states describing the verified bounds of the system evolution over a finite time horizon under a given set of commands. The framework naturally extends to systems whose continuous dynamics are subject to disturbances or other types of uncertainty.

We propose a three-step process for synthesis. The overall solution is summarized in Figure 3. Starting from an initial abstraction S_r, the first step involves performing synthesis on a specification that is realizable with respect to S_r. We next compute atomic controllers based on the synthesized finite-state machine. If any part of the low-level synthesis is not implementable, we update the discrete abstraction based on information extracted from the reachability computations. As reachability of nonlinear systems in general employs an expensive set of computations, our approach avoids unnec-

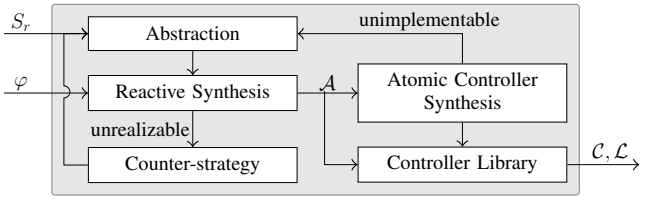


Fig. 3: Diagram of the procedure for adapting the discrete abstraction.

essary re-computation of reachability on the state machine transitions once the abstraction has been updated.

A. Atomic Controller Synthesis

We briefly review the process of synthesizing atomic controllers to implement a finite-state machine as in Definition 4. The procedure, denoted by $\mathsf{atomCtrl}(\mathcal{A})$, takes as its argument the discrete controller $\mathcal{A}$, and returns an atomic controller (κ_{ij}) and a reach set ($\mathcal{L}_{ij}$) for each transition $s' = \delta(s, \gamma^{\mathcal{X}}(s'))$ in $\mathcal{A}$.

While we may assume any reachability-based continuous controller synthesis technique to compute atomic controllers, in this work, we adopt the LQR trees approach of [19], adapted for reactive synthesis as explained in [7] and briefly summarized here. To satisfy Definition 4 for three transitions of $\mathcal{A}$, δ_i, $i = 1, \ldots, 3$, we require that the controllers for each transition be sequentially composable (there exists a valid controller for δ_2 and δ_3 once the execution of the controller for δ_1 completes), and that controllers for transitions that depend on the environment fulfill the stricter conditions for reactive composability (in a given region, the system can always invoke a controller to pursue either δ_2 or δ_3).

The computation of atomic controllers involves three steps: (1) generating a nominal trajectory that satisfies the boundary conditions of the transition, while avoiding obstacles and satisfying the transition constraints ($\Pi_{inv}(\pi_i, \pi_{aj})$); (2) computing a feedback control law to achieve trajectory stabilization; and (3) computing an invariant reach set given the trajectory and controllers subject to the same constraints. The reach set is computed by solving a constrained maximization problem (with mixed inequality and equality constraints) using a sums-of-squares technique [20] to solve for the quadratic Lyapunov functions.

B. Adaptation of Non-Deterministic Discrete Abstractions

During the process of constructing atomic controllers, we make necessary changes to the initial abstraction based on any partial results obtained from the reachability computations. We evaluate every transition that cannot be implemented to determine whether the successor is unreachable regardless of workspace constraints, or if the successor is reachable but violates the invariant. In the former case, we remove the failed successor from the abstraction. In the latter, we incorporate the reachable successors as (possibly non-deterministic) transitions. We formalize this process below.

1) Initialization of S_r: We require an initial discrete abstraction and extract a strategy if one exists. Tthere is no restriction to the form chosen for this initial guess except that the specification φ' is realizable on it. A reasonable choice is a topology graph where, for each region, there is a unique action that takes the robot to an adjacent region of its choice. In such an abstraction, the invariant set for each transition is assumed to be minimal; that is, $\Pi_{inv}(\pi_i, \pi_{aj}) = \{\pi_i, \pi_j\}$.

2) Reachability-Driven Updates to S_r: A transition $\delta_r(\pi_j, \pi_{ak})$ is said to have *failed* with respect to π_i if a controller κ_{jk} for a transition $\delta_r(\pi_j, \pi_{ak})$ composed with κ_{ij} for $\delta_r(\pi_i, \pi_{aj})$ produces trajectories that either do not reach the successor π_j, or do not stay in the given invariant set $\Pi_{inv}(\pi_j, \pi_{ak})$. In section IV-A, we briefly reviewed how we apply sums-of-squares optimization to find controllers that satisfy the constraints imposed by $\Pi_{inv}(\pi_j, \pi_{ak})$. Since our aim is to maximize workspace coverage, we find controllers that *maximize* the size of the reach sets subject to the invariance constraints and the constraints for composability. If we fail to satisfy these region-based constraints, we remove the constraints imposed by the set $\Pi_{inv}(\pi_j, \pi_{ak})$, and instead change our objective to finding the *minimal* reach set subject to the dynamics and the composability conditions of Definition 4. As we ultimately use the reach set to modify the invariant set, the minimization is to ensure that we minimize the size of this set (achieve a set with the fewest number of completion propositions), thus reducing undesirable non-determinism in the resulting abstraction.

In the case that construction of κ_{jk} has failed, we achieve an as-tight-as-possible fit of $\mathcal{L}_{jk}$ to $\mathcal{L}_{ij}(T_{ij})$ by solving the following minimization problem:

$$\begin{aligned} &\min vol[\mathcal{L}_{jk}] \\ \text{s.t. } &\mathcal{L}_{jk}(T_{jk}) \subset r_k, \\ &\mathcal{L}_{ij}(T_{ij}) \subseteq \mathcal{L}_{jk}(t), \mathcal{L}_{jk}(t) \subset W, \forall t \in [0, T_{jk}). \end{aligned} \tag{5}$$

where $vol[\mathcal{L}_{jk}]$ is defined as the volume of the reach set $\mathcal{L}_{jk}$, approximated as the sum of the computed volumes of $\mathcal{L}_{jk}$ at uniformly-spaced time instants. Note that minimizing $vol[\mathcal{L}_{jk}]$ in this way ensures that the set of successor regions π_k is kept as small as possible, while satisfying the constraints on composition and the workspace bounds.

Our goal now is to use these reachability results to compute appropriate modifications to the discrete abstraction. We do so by maintaining a one-step memory of the starting region for controller κ_{ij} (in this case π_i) whose composition with κ_{jk} resulted in failure. The invariant set for the failed transition ($\Pi_{inv}(\pi_j, \pi_{ak})$) is then *conditioned* on the starting region π_i of controller κ_{ij}, so that only those invariants corresponding to executions starting from r_i are adapted. That is, if the robot starts in r_i and activates the controller to r_j, but cannot progress from r_j to r_k without violating the invariants given in the abstraction, we modify the abstraction only for that path. On the other hand, if the robot starts at r_ℓ and activates a controller to r_j, and there is a controller taking the robot to r_k that satisfies the existing invariant, we do not modify the abstraction. Because we are not modifying the path r_ℓ, r_j, r_k, conditioning on the predecessor region produces an abstraction that is strictly less conservative than an unconditioned implementation where we modify that path.

We illustrate this point in Figure 4. Consider the construction of the controller κ_{52} in Figure 4a and 4b. In Figure 4a, in order for $\mathcal{L}_{45}$ to be sequenced with $\mathcal{L}_{52}$, we must extend the invariant ($\{\pi_5, \pi_2\}$) to include neighboring regions. In Figure 4b, on the other hand, we can sequence $\mathcal{L}_{75}$ with $\mathcal{L}_{52}$ with the existing invariant ($\{\pi_5, \pi_2\}$). Conditioning the updated invariant on r_4 allows the extended invariant to apply only for the sequence r_4, r_5, r_2. Note that we choose to condition only on the one previous region, as it is computationally less expensive than conditioning on a sequence of regions. This comes at the penalty of greater conservatism, since it allows for more environment behaviors.

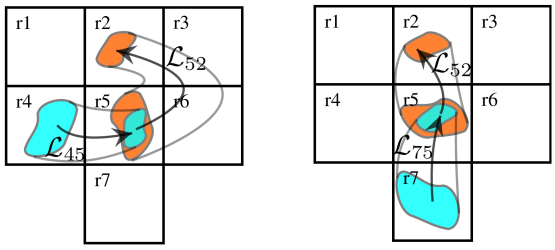


Fig. 4: Illustration of two controllers sequenced together with $\delta_r(\pi_5, \pi_{a2})$ as the final transition in both cases. (a) a case where the discrete abstraction is adapted to accommodate κ_{52} entering r_6 and r_3. (b) a case where the discrete abstraction does not need to be adapted.

To this end, we introduce a new invariant mapping function $\hat{\Pi}_{inv} : \mathcal{X}_c \times \mathcal{Y} \times \mathcal{X}_c \to 2^{\mathcal{X}_c}$, that produces a set of invariant regions for the transition $\delta_r(\pi_j, \pi_{ak})$, conditioned on the starting region π_i for the controller κ_{ij}. We write $\hat{\Pi}_{inv}(\pi_j, \pi_{ak} \mid \pi_i)$ as being the invariant for the transition $\delta(\pi_j, \pi_{ak})$ conditioned on π_i.[2]

We update the abstraction in different ways depending on the feasibility of the optimization problem in 5. If the problem is infeasible, this means that we cannot sequence together atomic controllers κ_{ij} and κ_{jk} while guaranteeing collision-free trajectories. In this case, we remove the encoding of the transition $\delta_r(\pi_j, \pi_{ak})$ from φ'. If the problem is feasible, then there exists a sequence of controllers that produce trajectories that do not leave the workspace, but pass through regions other than those in $\Pi_{inv}(\pi_j, \pi_{ak})$. In this case, we compute the updated invariant mapping $\hat{\Pi}_{inv}(\pi_j, \pi_{ak} \mid \pi_i)$ to be:

$$\hat{\Pi}_{inv}(\pi_j, \pi_{ak} \mid \pi_i) = \{\pi_\ell \mid \forall r_\ell \subset \mathcal{R}, \mathcal{L}_{jk} \cap r_\ell \neq \emptyset, \mathcal{L}_{ij}(T_{ij}) \subseteq \mathcal{L}_{jk}\}. \quad (6)$$

Example 1. Consider the two reach sets shown in Figure 4a, and assume S_r corresponds to the topology graph. Observe that the set $\mathcal{L}_{52}$ violates the topology graph constraint $r_5 \cup r_2$ ($\Pi_{inv}(\pi_5, \pi_{a2}) = \{\pi_5, \pi_2\}$), but all trajectories from $\mathcal{L}_{45}(T_{45})$ are able to reach r_2 when not subject to this constraint. In this case, the minimal invariant that allows $\mathcal{L}_{52}$ to be sequentially composable with $\mathcal{L}_{45}$, is $\{\pi_5, \pi_6, \pi_2, \pi_3\}$. We therefore obtain the update $\hat{\Pi}_{inv}(\pi_5, \pi_{a2} \mid \pi_4) = \{\pi_5, \pi_6, \pi_2, \pi_3\}$.

3) Modifying the LTL Encoding: Whenever $\mathcal{A}$ is unimplementable, $\mathsf{atomCtrl}(\mathcal{A})$ returns the completion and activation propositions corresponding to each of the failed transitions. These are collected in the set $P_{fail} \subseteq \mathcal{X}_c \times \mathcal{X}_c \times \mathcal{Y}$, consisting of the predecessor π_i, the current region π_j and the activation π_{ak}. For failed transitions (π_j, π_{ak}) originating from initial states in $\mathcal{A}$, we store $(\texttt{True}, \pi_j, \pi_{ak})$ in P_{fail} since π_i is undefined initially.

In the case where, for some $(\pi_i, \pi_j, \pi_{ak}) \in P_{fail}$, the optimization problem (5) is *feasible*, we incorporate the updated invariant $\hat{\Pi}_{inv}(\pi_j, \pi_{ak} \mid \pi_i)$ into the abstraction. Let $y_{mi} \in \mathcal{Y}$ denote a memory proposition for region π_i that keeps memory of when the predecessor was π_i . We encode this behavior by adding to ψ^r_{tc} (2) the following set of conjuncts:

$$\bigwedge_{(\pi_i,\pi_j,\pi_{ak}) \in P_{fail}} \Bigg(\square \left((\bigcirc \pi_i \vee y_{mi}) \wedge \bigcirc(\pi_i \vee \pi_j) \iff \bigcirc y_{mi} \right) \wedge \square \left(y_{mi} \wedge \pi_j \wedge \pi_{ak} \implies \bigvee_{\pi_\ell \in \hat{\Pi}_{inv}(\pi_j, \pi_{ak} \mid \pi_i)} \bigcirc \pi_\ell \right) \Bigg). \quad (7)$$

According to the first conjunct, y_{mi} is set upon entering π_i and reset upon exiting $\pi_i \vee \pi_j$. The second conjunct updates the invariant regions that may be visited given the current action and current and past completions.

In the case where, for some $(\pi_i, \pi_j, \pi_{ak}) \in P_{fail}$, (5) is *infeasible*, we remove the transition entirely, since the controller κ_{ij} is blocking (it cannot reach π_k). We update the liveness condition in (4) to account for this, by removing the possibility of completions occurring when the regions π_i and π_j are visited in order, and the action π_{ak} is applied:

$$\psi^r_g = \square \diamondsuit \bigvee_{\substack{\pi_j \in \mathcal{X}_c, y_{mi}, \pi_{ak} \in \mathcal{Y} \\ (\pi_i, \pi_j, \pi_{ak}) \notin P_{fail}}} (y_{mi} \wedge \pi_j \wedge \pi_{ak} \wedge \bigcirc(\pi_k \vee \neg \pi_{ak})) \quad (8)$$

Note that (8) resembles (4), aside from the added conditions on the disjunct: $(\pi_i, \pi_j, \pi_{ak}) \notin P_{fail}$.

Example 2. Consider again Example 1 and Figure 4a. In this case, $P_{fail} = \{(\pi_4, \pi_5, \pi_{a2})\}$. We modify ψ^r_{tc} by applying the following as an additional set of conjuncts:

$$\square \left((\bigcirc \pi_4 \vee y_{m4}) \wedge \bigcirc(\pi_4 \vee \pi_5) \iff \bigcirc y_{m4} \right)$$
$$\wedge \square \left(y_{m4} \wedge \pi_5 \wedge \pi_{a2} \implies (\bigcirc \pi_5 \vee \bigcirc \pi_6 \vee \bigcirc \pi_3 \vee \bigcirc \pi_2) \right).$$

Algorithm 1 Synthesizing controllers for a specification φ under an initial abstraction S_r.

$\varphi' \leftarrow$ LTL encoding of S_r (2), (3), (4)
$\mathcal{A} \leftarrow \mathsf{realizable}(\varphi')$
if realizable **then**
  $(\mathcal{C}, \mathcal{L}, P_{fail}) \leftarrow \mathsf{atomCtrl}(\mathcal{A})$
  while realizable and not implementable **do**
    for all $(\pi_i, \pi_j, \pi_{ak}) \in P_{fail}$ s.t. $\hat{\Pi}_{inv}(\pi_j, \pi_{ak} \mid \pi_i)$ is undefined **do**
      $\hat{\Pi}_{inv} \leftarrow$ (6)
      feasible $\leftarrow$ solution to problem (5)
      if feasible **then**
        $\varphi' \leftarrow$ (7)
      else
        $\varphi' \leftarrow$ (8)
      end if
    end for
    $\mathcal{A} \leftarrow \mathsf{realizable}(\varphi')$
    $(\mathcal{C}, \mathcal{L}, P_{fail}) \leftarrow \mathsf{atomCtrl}(\mathcal{A})$
    Compute $\hat{\Pi}_{inv}$
  end while
  if realizable and implementable **then**
    return $\mathcal{A}$, $\mathcal{C}$, $\mathcal{L}$
  end if
end if
return failure

[2]Note that we can generalize $\hat{\Pi}_{inv}$ to account for N previous regions as $\hat{\Pi}_{inv}(\pi_j, \pi_{ak} \mid \pi_{i1}, \pi_{i2}, \ldots, \pi_{iN})$, however for complexity reasons we consider only $N = 1$.

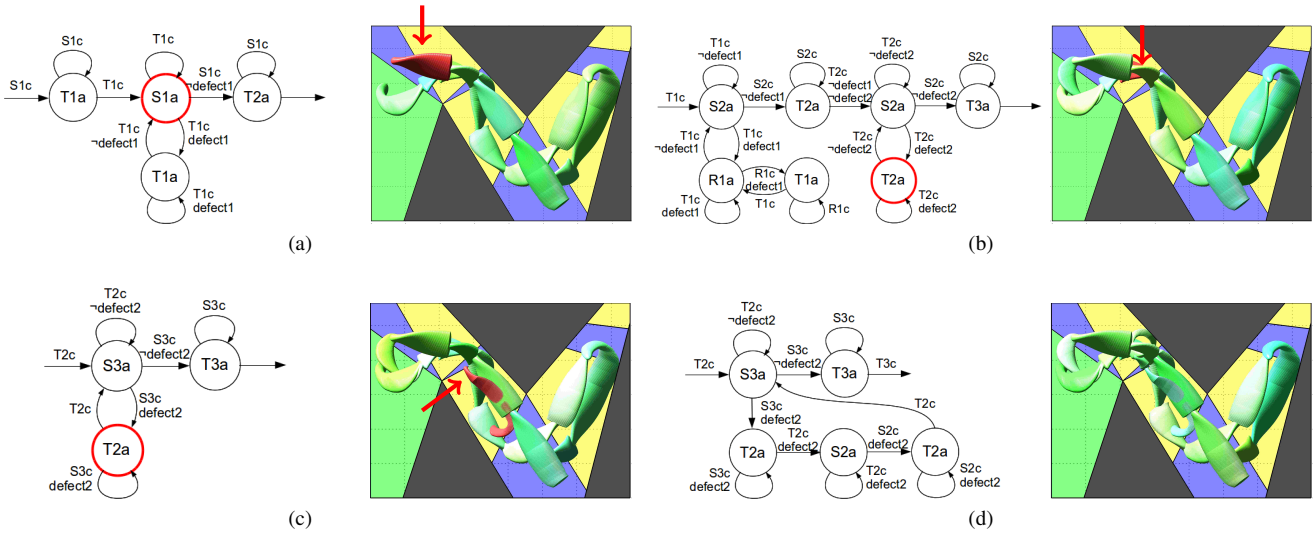


Fig. 5: Atomic controller synthesis results for the first three iterations of the algorithm. In subfigures (a)– (c), a partial finite-state machine is shown, along with a partial set of reach sets for each controller. The states indicated in red correspond to transitions in $\mathcal{A}$ that cannot be implemented at the low level (π_{ak} in P_{fail}), and the reach set highlighted red indicates the associated reach set $\mathcal{L}_{\pi_i,\pi_j}$ for $(\pi_i, \pi_j, \pi_{ak}) = P_{fail}$. (d) shows the implementation results after the first three iterations.

C. Iterative Procedure

We now discuss the main algorithm. The function $\mathsf{realizable}(\varphi')$ checks for realizability of φ' on the initial abstraction S_r, using the algorithm in [4] and returns $\mathcal{A}$ if it is realizable. The abstraction is updated iteratively (lines 5–18), starting by synthesizing a strategy, then computing the atomic controllers, and finally adapting the abstraction. The adaptation steps take place in lines 6–14; in line 7, the invariant is relaxed to include additional regions according to the reach set computations. The result is *implementable* if controllers are found; otherwise the abstraction is adapted accordingly. The process is repeated until either φ is both realizable and implementable, or the specification becomes unrealizable. At each iteration, we re-start $\mathsf{atomCtrl}(\mathcal{A})$ re-using any partial results of the implementation saved from the previous steps. In so doing, we avoid the potentially costly step of re-computing existing sets of verified controllers.

Given the number of possible environment proposition combinations, $n = 2^{|\mathcal{X}\setminus\mathcal{X}_c|}$, and the number of region propositions, $m = |\mathcal{Y}|$, we compute invariants for m regions up to n^2 times under the assumption of one-step memory, and add at most m memory propositions to the set $\mathcal{Y}$. The complexity of the algorithm is therefore $O(mn^2)$. This is a very conservative upper-bound as, in our experience, far fewer steps are needed. We note that the approach is sound but not complete since, for nonlinear systems, reach sets are computed using conservative overapproximations to the true bounds of the system.

V. Example

We have implemented this approach in a Matlab routine that employs the slugs synthesis tool[3] to perform realizability checking. We demonstrate our results in an autonomous inspection scenario that takes place in the workspace map of Figure 6. The specification requires the robot to patrol the targets $T_1, \ldots, T_4$. If it sees a defect, it stops patrolling and visits the target where the defect was sensed; defects are treated as uncontrolled environment variables. The full specification is listed in 1.

Specification 1 Inspect the structure, marking any found defects by remaining there.

1: Start in R_1.
2: If no defects, visit targets T_1, T_2, T_3, T_4.
3: If $defect_i$, visit T_i. (mark the target)
4: If $defect_i$, never enter $\bigvee_{j,j\neq i} T_j$. (disambiguation)
5: Only $defect_i$ if in T_i or its immediate neighbors. (no false alarms)
6: $defect_i \iff \neg \bigwedge_{j,j\neq i} defect_j$. (mutual exclusion)

A three-state robot with dynamics described by a kinematic unicycle model is used to implement this task. The unicycle model consists of three continuous states (x, y, θ), two Cartesian displacements and an orientation angle. The robot has the ability to turn but with limited turning radius and is assumed to move at a fixed forward velocity. The robot's command input is its angular rate, ω.

Initializing S_r with a topology graph, finding implementable controllers required 7 calls to $\mathsf{realizable}(\varphi')$ to check realizability, and took 57 minutes to compute on a laptop with an Intel Core i7 2.8GHz processor and 8GB of RAM. Atomic controllers are constructed starting at the initial condition R_1, traversing each transition in the finite-state machine until each transition has been addressed. The first encountered unimplementable transition occurred in the transition $\delta_r(T_1, T_{a1})$, where the finite-state machine requires a self-loop transition at T_1 (state $Sa1$ labeled red in Figure 5a). The minimization problem in (5) was feasible in this case, and a sequentially-composable controller was computed

[3]https://github.com/LTLMoP/slugs

in which the robot is required to pass through R_1 in order to reach T_1. In this case, the propositions $P_{fail} = (S_1, T_1, S_{a1})$ (the reach set for $\delta_r(S_1, T_1)$ is labeled red) and the invariant set is $\Pi_{inv}(T_1, S_{a1} \mid S_1) = \{R_1, T_1, S_1\}$. The abstraction is updated by introducing safety statements to ψ^r_{tc} that require both T_1 and R_1 be included in the invariant set for the transition $\delta_r(T_1, T_{a1})$ with respect to S_1. Two separate iterations of Algorithm 1 are shown in Figure 5b and Figure 5c. In both cases, the abstraction is adapted in the neighborhood of T_2, due to the small size of the region. Similar to the first case, these two cases are adapted by adding neighboring regions as successors at T_2 (states labeled red in the figure). With the fully-adapted abstraction, the implementation produces the trajectory shown in Figure 6.

The adaptation in this example required 5 additional propositions. The time required to synthesize the specification every time a memory proposition was added is shown in Table I. Although the propositions introduce complexity in the GR(1) synthesis process, this increase is small compared with the overall time required to construct atomic controllers.

TABLE I: Synthesis time versus number of memory propositions.

# mem props	0	1	2	3	4	5
time (sec)	< 1.0	1.5	4.4	9.3	31.1	80.3

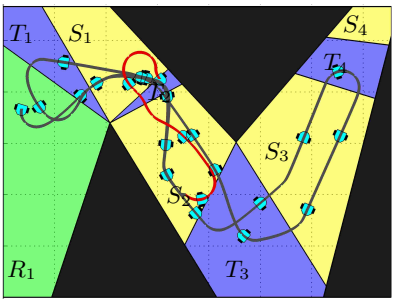


Fig. 6: An execution for the case where a defect is found in T_2. The red portion of the trajectory indicates when $defect_2 =$`True`.

VI. Conclusions

In this paper, we have presented a synthesis approach for dynamical systems that takes a specification and a simplified robot abstraction and synthesizes a controller that implements the specification on a robot with complex dynamics. The approach allows a user to supply a suitable guess at a discrete abstraction for the robot, updating this abstraction in a local fashion if parts of the resulting strategy are unimplementable. Our approach re-uses, to the extent possible, the set of atomic controllers that have already been constructed.

There are several opportunities for future work, including providing feedback to the user when specifications become unrealizable as a result of alterations made to the abstraction. We plan to further explore how the choice of the initial abstraction impacts the synthesis results, and its implications on eventually realizing the specification using our method. We also plan to extend our approach to controller synthesis for continuous dynamics that are learned on-the-fly. Future work will also be directed toward experimental testing, and empirical comparison with other methods for abstraction and synthesis.

References

[1] E. Aydin Gol, M. Lazar, and C. Belta. Language-guided controller synthesis for discrete-time linear systems. In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, pages 95–104. ACM, 2012.

[2] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation (ICRA 2010)*, pages 2689–2696. IEEE, 2010.

[3] A. Bhatia, M.R. Maly, L.E. Kavraki, and M.Y. Vardi. Motion planning with complex goals. *Robotics Automation Magazine, IEEE*, 18(3):55 –64, sept. 2011.

[4] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

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

[6] E.M. Clarke. Counterexample-guided abstraction refinement. In *10th International Symposium on Temporal Representation and Reasoning / 4th International Conference on Temporal Logic (TIME-ICTL 2003)*, page 7, 2003.

[7] J.A. DeCastro and H. Kress-Gazit. Synthesis of nonlinear continuous controllers for verifiably-correct high-level, reactive behaviors. *International Journal of Robotics Research*. Accepted.

[8] J. Ding, J. Gillula, H. Huang, M.P. Vitus, W. Zhang, and C.J. Tomlin. Hybrid systems in robotics: Toward reachability-based controller design. *IEEE Robotics & Automation Magazine*, 18(3):33 – 43, Sept. 2011.

[9] G.E. Fainekos, S.G. Loizou, and G.J. Pappas. Translating temporal logic to controller specifications. In *Proc. of the 45th IEEE Conf. on Decision and Control (CDC 2006)*, pages 899–904, 2006.

[10] H. Kress-Gazit, D.C. Conner, H. Choset, A.A. Rizzi, and G.J. Pappas. Courteous cars. *IEEE Robot. Automat. Mag.*, 15(1):30–38, 2008.

[11] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[12] J. Liu and N. Ozay. Abstraction, discretization, and robustness in temporal logic control of dynamical systems. In *Proc. of the 17th Int. Conf. on Hybrid Systems: Computation and Control (HSCC'14)*, 2014.

[13] J. Liu, N. Ozay, U. Topcu, and R.M. Murray. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Trans. Automat. Contr.*, 58(7):1771–1785, 2013.

[14] M.R. Maly, M. Lahijanian, L.E. Kavraki, H. Kress-Gazit, and M.Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 353–362, Philadelphia, PA, USA, 08/04/2013 2013. ACM, ACM.

[15] P. Nilsson and N. Ozay. Incremental synthesis of switching protocols via abstraction refinement. In *Proc. 53rd IEEE Conference on Decision and Control (CDC) 2014*, 2014.

[16] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Proc. of the 4th Int. Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, pages 477–492, 2004.

[17] V. Raman, N. Piterman, C. Finucane, and H. Kress-Gazit. Timing semantics for abstraction and execution of synthesized high-level robot control. *IEEE Transactions on Robotics*, 2015.

[18] V. Raman, N. Piterman, and H. Kress-Gazit. Provably correct continuous control for high-level robot behaviors with actions of arbitrary execution durations. In *IEEE International Conference on Robotics and Automation*, pages 4075–4081, Karlsruhe, Germany, 2013.

[19] R. Tedrake, I.R. Manchester, M. Tobenkin, and J.W. Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *I. J. Robotic Res.*, 29(8):1038–1052, 2010.

[20] M.M. Tobenkin, I.R. Manchester, and R. Tedrake. Invariant funnels around trajectories using sum-of-squares programming. In *Proc. of the 18th IFAC World Congress*, 2011.

[21] E.M. Wolff, U. Topcu, and R.M. Murray. Automaton-guided controller synthesis for nonlinear systems with temporal logic. In *IROS*, pages 4332–4339. IEEE, 2013.

[22] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon control for temporal logic specifications. In *Proc. of the 13th Int. Conf. on Hybrid Systems: Computation and Control (HSCC'10)*, 2010.