ANNUAL REVIEWS

Annual Review of Control, Robotics, and Autonomous Systems

Synthesis for Robots: Guarantees and Feedback for Robot Behavior

Hadas Kress-Gazit,[1] Morteza Lahijanian,[2] and Vasumathi Raman[3]

[1]Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, New York 14853, USA; email: hadaskg@cornell.edu

[2]Department of Computer Science, University of Oxford, Oxford OX1 3QD, United Kingdom; email: morteza.lahijanian@cs.ox.ac.uk

[3]San Francisco, California 94103, USA; email: vasu@cds.caltech.edu

Annu. Rev. Control Robot. Auton. Syst. 2018. 1:211–36

First published as a Review in Advance on January 19, 2018

The *Annual Review of Control, Robotics, and Autonomous Systems* is online at control.annualreviews.org

https://doi.org/10.1146/annurev-control-060117-104838

Keywords

robotics, formal methods, synthesis, control, high-level specifications

Abstract

Robot control for tasks such as moving around obstacles or grasping objects has advanced significantly in the last few decades. However, controlling robots to perform complex tasks is still accomplished largely by highly trained programmers in a manual, time-consuming, and error-prone process that is typically validated only through extensive testing. Formal methods are mathematical techniques for reasoning about systems, their requirements, and their guarantees. Formal synthesis for robotics refers to frameworks for specifying tasks in a mathematically precise language and automatically transforming these specifications into correct-by-construction robot controllers or into a proof that the task cannot be done. Synthesis allows users to reason about the task specification rather than its implementation, reduces implementation error, and provides behavioral guarantees for the resulting controller. This article reviews the current state of formal synthesis for robotics and surveys the landscape of abstractions, specifications, and synthesis algorithms that enable it.

1. INTRODUCTION

Formal methods are mathematical tools and techniques used in several engineering domains to reason about systems, their requirements, and their guarantees (1). Typically, formal methods address two questions: verification (given a set of requirements or specifications and a system model, does the system satisfy the specifications?) and synthesis (given a set of specifications, can one generate a system that is correct by construction, i.e., built in a way that is guaranteed to satisfy the requirements?).

The state of the art in robot control for tasks such as moving around obstacles or grasping objects has advanced significantly in the last few decades through the development of motion planners and learning algorithms. However, getting robots to perform complex tasks such as completing the DARPA Robotics Challenge (2) is still accomplished largely by a team of highly trained programmers who manually compose the different system components together. This manual process is time consuming, error prone, and typically validated only through extensive testing.

Formal synthesis for robotics provides a framework for specifying complex robot tasks in a mathematically precise language and automatically transforming these specifications into correct-by-construction robot controllers when feasible. This approach allows a user to reason about the task specification rather than the actual implementation, reduces implementation errors, and provides guarantees for the overall robot behavior. Furthermore, the formal description of the task enables feedback to be provided regarding the specifications themselves, such as whether they can be implemented by a physical robot in the possibly unknown environment. The synthesis approach to robot control, depicted in **Figure 1**, takes as input a specification and a model of the robot, potentially also with the initial state of the robot, a model of the environment, and/or a cost function, and outputs either a controller or a proof that the specification is not feasible (i.e., is unrealizable). Roughly speaking, the synthesis techniques can be grouped into three types:

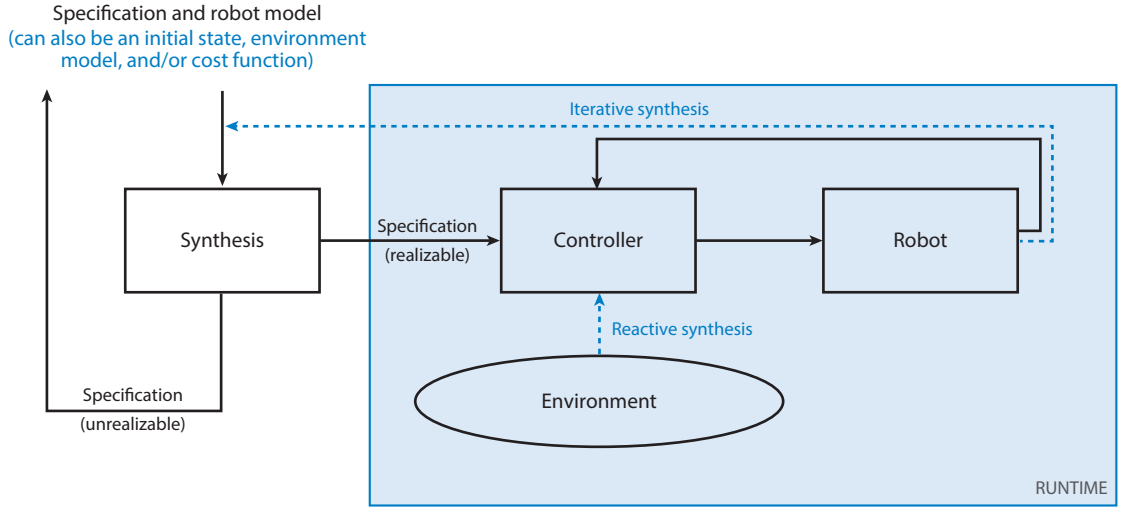

Figure 1

Synthesis for robot control. The input here is a specification and a robot model; some approaches also take as input the initial state of the robot, a model of the environment, and/or a cost function. The result of the synthesis algorithm is either a controller to be executed by the robot or a proof that the specification is not feasible (i.e., is unrealizable). The black elements in the figure are common to all synthesis approaches, while the blue elements are present only in some of them.

- Open loop (nonreactive): Given a robot model and a specification, find a sequence of states or actions that will guarantee that the robot satisfies the specification. In this approach, the environment is static and is typically not modeled.
- Iterative: Given a robot model, a prediction of the environment at each iteration, and a specification, find a sequence of states or actions at each iteration such that the robot satisfies the specification over the full execution horizon. Synthesis is performed repeatedly, either periodically (in a receding horizon or model predictive control manner) or when the expected environment changes.
- Reactive: Given a robot model, an environment model, and a specification, find a strategy (i.e., a function from states to actions or other states) that will guarantee that the robot satisfies the specification under any modeled environment behavior. The environment is typically modeled as uncertain or adversarial.

As described in Section 3, researchers have explored a wide range of specification formalisms with different expressive power. These include discrete temporal logics, which are defined over symbolic abstractions of the continuous system; probabilistic temporal logics, where the task definition includes constraints on the probability of success; and metric and signal temporal logics, which can express constraints on continuous time and state, respectively.

This article reviews the state of the art in formal synthesis of controllers for robots from temporal logic. It discusses the algorithms used to transform the continuous problem of robot motion and action to and from the symbolic structures used for synthesis (Section 2), the specification formalisms used to capture requirements for robot behavior and assumptions about the environment (Section 3), and algorithmic approaches to synthesis (Section 4). It focuses on single-robot systems and temporal logic specifications; multirobot systems (e.g., 3–13) and recent approaches such as synthesis through satisfiability modulo theories (e.g., 14–16) are beyond the scope of the review.

1.1. Guarantees and Feedback

One of the main advantages of a formal synthesis framework for controlling robots is the ability to provide both guarantees and feedback regarding task feasibility.

1.1.1. Guarantees. The synthesis approach takes a set of specifications and a system model and generates a controller that achieves the specifications, if one exists. The algorithms are sound (if a controller is found, it is correct), and most are complete (if a controller exists, it will be found). Correctness here means that the system, at the level of abstraction of the model, will satisfy its specification in any modeled environment. This does not mean that a robot will never fail; the fidelity of the model with respect to reality will govern the success of the actual physical execution. The correct-by-construction guarantees with respect to the specification and abstraction of the synthesized controller, together with the ability to refine the abstraction, eliminate human error in implementation and are a strong indicator of success, especially compared with the manual composition of controllers.

1.1.2. Feedback and suggestions. Owing to the formal problem description, if synthesis fails and no controller is produced, then the specification cannot be fully realized by the models of the system and the environment. This means that there exists a counterexample that shows under what conditions the robot will fail. Leveraging these counterexamples, synthesis frameworks are able to produce explanations for what can go wrong and suggestions for how to modify the task to make it achievable. Furthermore, they can provide feedback about inconsistencies, redundancies,

and vacuity in the specifications themselves, which are often written by humans and are therefore error prone.

Methods for enabling feedback on such unrealizable specifications cover logical inconsistencies and environment behaviors that can prevent robot success at the specified abstraction level (17, 18), physical constraints that prohibit the robot from following the symbolic solution (19), and specifications that are vacuous or tautological (17, 20), i.e., where a controller can be created that may not do anything. In the context of suggesting changes, References 21–26 explore minimum-distance revisions for linear temporal logic (LTL) and automata-based specifications. References 27 and 28 explore methods for generating additional symbolic environment assumptions that would make the specification realizable. Automatic revisions to the abstractions based on the dynamics of the robot are presented in References 29 and 30, and suggested revisions owing to probabilistic analysis of a synthesized controller are discussed in References 26 and 31. Specification diagnosis and revision have also been explored for optimization-based synthesis approaches that do not involve a discrete abstraction (32).

1.2. Relation to Other Communities

Synthesis, as discussed in this review, is used to automate the creation of robot controllers from high-level specifications, thereby enabling users to reason about properties of robot behavior and automatically generate a correct implementation of the behavior for the physical system. The process of automating high-level behavior is the major focus of related communities, most notably the artificial intelligence planning community and the discrete event systems community.

In artificial intelligence, the planning problem is typically represented as a set of actions, each with preconditions and postconditions, an initial state, and a goal state expressed using the Planning Domain Definition Language (PDDL) (33) or one of its variants. Generally, planning algorithms search for a sequence of actions that will lead the system from the initial state to the goal state. Variants that are closer to the work described in this review are those that handle temporally extended goals (e.g., 34, 35), i.e., goals that are more complex than a single state; universal planners (e.g., 36), which synthesize reaction rules for possible environment behaviors; and contingency planners (e.g., 37, 38), which create branching plans where the system makes a decision based on the environment. Some work has also explored temporal logic for specifying goals and leveraged model-checking techniques for planning (39, 40). The differences between synthesis approaches and those pursued in the planning community span the way the problem is formulated, the complexity of the algorithms, the expressiveness of the specifications and system models, and the type of feedback that is possible when the synthesis or planning problem can and cannot be solved.

In discrete event systems, the system (plant) is a transition system with states and transitions. The main difference between the system model in discrete event systems and in synthesis is that in the former, the transitions are separated into controlled and uncontrolled transitions. The main problem addressed by the discrete event systems community is finding a supervisory controller that chooses which controlled transitions to take so that the system achieves a high-level behavior. Reference 41 presents a comparison of supervisory control in discrete event systems and reactive synthesis.

1.3. Example

The following example illustrates the concepts discussed in this review. Consider a mobile robot moving in the workspace depicted in **Figure 2**. This workspace contains areas of interest, such as

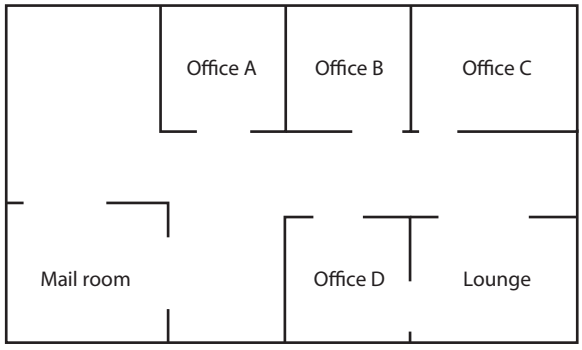

Figure 2

An example workspace for mail delivery by a robot.

a mail room and offices, that may be used as part of the specification describing the robot's desired behavior.

In addition to actuation, the robot is equipped with exteroceptive sensors, such as cameras and range finders. We assume that the rich information from these sensors is abstracted into discrete symbols, such as "person detected" or "alarm is on," that can be either true or false. These symbols can be the output of classifiers or other perception algorithms. The task used throughout this review is a mail-delivery task, where the robot is instructed to deliver letters and packages to rooms and/or people.

2. ABSTRACTIONS: SYSTEM REPRESENTATIONS

The synthesis algorithms described in this review require abstractions of the task, the environment, and the physical robot behavior (i.e., its dynamics). These abstractions map the physical, continuous problem of robot motion and action into sets of symbols that can be reasoned about and mapped back to sensing and control for the robot. This section describes robot dynamics models (Section 2.1), symbolic structures used in the synthesis algorithms (Section 2.2), and techniques for mapping the physical to the symbolic and back (Section 2.3).

2.1. Robot Models

All approaches to robot controller synthesis assume a robot dynamics model; the model can be either continuous or discrete time and with or without disturbances. The continuous and logical states of the robot are denoted with $x \in X \subseteq (\mathbb{R}^{n_c} \times \{0,1\}^{n_l})$, the continuous and logical control inputs with $u \in U \subseteq (\mathbb{R}^{m_c} \times \{0,1\}^{m_l})$, and the (possibly adversarial) external inputs in the form of noise or disturbances $w \in W \subseteq (\mathbb{R}^{e_c} \times \{0,1\}^{e_l})$. The system model for each approach is one of

$$\dot{x} = f(x,u), \quad \dot{x} = f(x,u,w),$$

and is in some cases assumed to admit a discrete-time approximation of the form

$$x(t_{k+1}) = f_d(x(t_k), u(t_k)), \quad x(t_{k+1}) = f_d(x(t_k), u(t_k), w(t_k)),$$

where for all $k > 0$, $t_{k+1} - t_k = \Delta t$.

Trajectory ξ: the execution of the system dynamics model

A system trajectory ξ is an execution of the system dynamics from an initial state x_0. In the discrete-time model, $\xi = (x_0 u_0 w_0)(x_1 u_1 w_1)(x_2 u_2 w_2)\ldots$ becomes a sequence of states, control actions, and external inputs.

2.2. Symbolic Representations

Path ω: a path in symbolic structure

Trace (word) σ: a sequence of labels

$Path_s$: the set of all infinite paths starting at state s

$Path_s^{fin}$: the set of all finite paths starting at state s

Control policy μ: a function mapping finite paths to actions

The synthesis techniques covered in this review are based on deterministic, nondeterministic, and probabilistic discrete structures. Each structure is assumed to be defined with respect to a set of symbols, referred to as the set of atomic propositions AP.

2.2.1. Kripke structure. Given a set of atomic propositions AP, a Kripke structure over AP is a tuple $\mathcal{K} = (S, S_0, R, L)$ (42), where

- S is a finite set of states;
- $S_0 \subseteq S$ is the set of initial states;
- $R \subseteq S \times S$ is a transition relation where for all $s \in S$ there exists a state $s' \in S$ such that $(s, s') \in R$; and
- $L : S \to 2^{AP}$ is the labeling function such that $L(s) \subseteq AP$ is the set of atomic propositions that are true in state s.

A path ω in $\mathcal{K}$ is an infinite sequence $\omega = \omega_0 \to \omega_1 \to \omega_2 \cdots$, where $\omega_0 \in S_0$, $\omega_i \in S$, and $(\omega_i, \omega_{i+1}) \in R$ for all $i \geqslant 0$. Given a path ω, a trace (word) σ over ω is defined as $\sigma = L(\omega_0)L(\omega_1)L(\omega_2)\cdots$, where $L(\omega_i) \in 2^{AP}$ is the label of state ω_i.

2.2.2. Labeled Markov decision processes. A labeled Markov decision process (MDP) (adapted from 43) is a tuple $\mathcal{M} = (S, s_0, Act, Steps, AP, L)$ where

- S is a finite set of states;
- $s_0 \in S$ is the initial state;
- $Act = \bigcup_{s \in S} A(s)$ is the set of actions, where $A(s)$ denotes the set of available actions at state s;
- $Steps : S \times Act \to Dist(S)$ is a (partial) probabilistic transition function that maps each state–action pair (s, a), $s \in S$, and $a \in A(s)$ to a discrete probability distribution over S;
- AP is a set of atomic propositions used to label the states; and
- $L : S \to 2^{AP}$ is the labeling function such that $L(s) \subseteq AP$ is the set of atomic propositions that are true in state s.

A path ω in $\mathcal{M}$ is a sequence $\omega = \omega_0 \xrightarrow{a_1} \omega_1 \xrightarrow{a_2} \omega_2 \cdots$, where $\omega_0 = s_0$, and for all $i \geqslant 0$, $\omega_i \in S$, $a_{i+1} \in A(\omega_i)$, and $Steps(\omega_i, a_{i+1})(\omega_{i+1}) > 0$; ω is used to indicate an infinite path, ω^{fin} to denote a finite path, and $last(\omega^{fin})$ to denote the last state of a finite path. $Path_s$ and $Path_s^{fin}$ are used to indicate the set of all infinite paths ω and the set of all finite paths ω^{fin} starting at state s, respectively.

A control policy μ is a function mapping finite paths $\omega^{fin} \in Path_s^{fin}$ of $\mathcal{M}$ to an action $a \in Act$ such that $a \in A(last(\omega^{fin}))$. If policy μ depends only on $last(\omega^{fin})$, then it is history independent and is called a stationary policy.

2.3. Physical Interpretation

As mentioned in Section 2.2, synthesis is performed on symbolic structures; however, the resulting controller is implemented on the physical system. Crucial to the success of the robot's behavior is its ability to continuously implement all of the symbolic transitions in the controller. This is formally defined as a simulation relation (44) where one system can mimic all of the behaviors of the second system. For robots, ideally, the continuous physical system simulates the symbolic one.

The following sections describe methods and algorithms for mapping states and controls of the continuous system to labels and transitions of the symbolic structures. These processes of abstraction create symbols and assign physical meaning to them. This section describes the creation

of abstractions relating to the robot motion and action in the workspace. To abstract sensor information or (more generally) the environment state, one can use classifiers or other perception algorithms, which are beyond the scope of this review.

The abstraction algorithms fall into three categories: partitions (Section 2.3.1), where the labels are mutually exclusive and the continuous system simulates the symbolic; motion primitives (Section 2.3.2), where the labels may overlap and the continuous system simulates the symbolic; and motion planners (Section 2.3.3), where the labels are typically mutually exclusive but the transitions are probabilistically complete, meaning that the continuous system might not be able to simulate the symbolic one.

For the following, consider the workspace of the robot $\mathcal{W}$ that contains nonoverlapping regions of interest Reg_i such that $Reg_i \cap Reg_j = \emptyset$ for all $i \neq j$. The regions do not necessarily cover the entire workspace.

2.3.1. Partitions. In this approach, the continuous state space of the robot $X \subseteq \mathbb{R}^{n_c}$ is partitioned into a set $\{r_i\}$ such that $r_i \subseteq X$, $\cup_i r_i = X$, and $r_i \cap r_j = \emptyset$ for all $i \neq j$. Overloading the notation, we use the symbol r_i as a label in the symbolic structure to mean that when a node is labeled with r_i, the physical system's state $x \in r_i \subseteq X$. Because of the partition, the r_i symbols are mutually exclusive; that is, no more than one symbol can be true at any time. Transitions in the discrete structure correspond to possible actions of the system and are related to the adjacency relationship of the cells in the partition. If a transition exists, then the cells are adjacent and the system can move from one cell to the other, but the reverse does not hold; that is, if cells are adjacent, there may not be a transition in the symbolic structure. **Figure 3** depicts a possible partition for a subset of the workspace of the mail-delivery example (**Figure 3*a***, subpanel *i*) and the corresponding symbolic structure (**Figure 3*b***, subpanel *i*).

There are different approaches to creating such partitions depending on the dynamics of the robot. For a holonomic robot $\dot{x} = u$ moving in a two- or three-dimensional workspace partitioned into polytopes, approaches such as those described in References 45–47 create vector fields that are used as a feedback controller to drive the robot from any state in a region to an adjacent region. There, the workspace regions Reg_i and a convex decomposition of the rest of the workspace $\mathcal{W}$ are the cells and labels. By considering points on boundaries as the goal set, one can use other potential field-based controllers, such as navigation functions (48), to create the symbolic structure. For multirobot tasks, similar decompositions can be created (49).

For robots with more complex, possibly nonlinear dynamics, other approaches discretize the n_c-dimensional state space of the system together with the set of control inputs into a high-dimensional grid and create a nondeterministic structure that takes into account the effects of discretization. These approaches typically compute, for each cell in the grid, an overapproximation of the set of cells reachable under a control action. Different techniques exist that vary in their assumptions regarding the underlying dynamics and the fidelity of the abstraction with respect to the full model (e.g., 50, 51). Reference 52 discusses the robustness of such abstractions to phenomena such as delays, measurement errors, and model uncertainties. Based on these ideas, the hybrid systems community has created different tools that automatically create the abstraction given the system model and the environment (53–55).

2.3.2. Motion primitives. Similar to the partition approach, the physical meaning of the symbols in the abstraction using motion primitives is related to regions of the state space. The main differences are that the state space is no longer divided into a grid, the regions representing motion primitives are usually not disjoint, and the set of regions does not have to cover the whole state space. Formally, the state space of the robot $X \subseteq \mathbb{R}^{n_c}$ contains a set $\{r_i\}$ such that $r_i \subseteq X$.

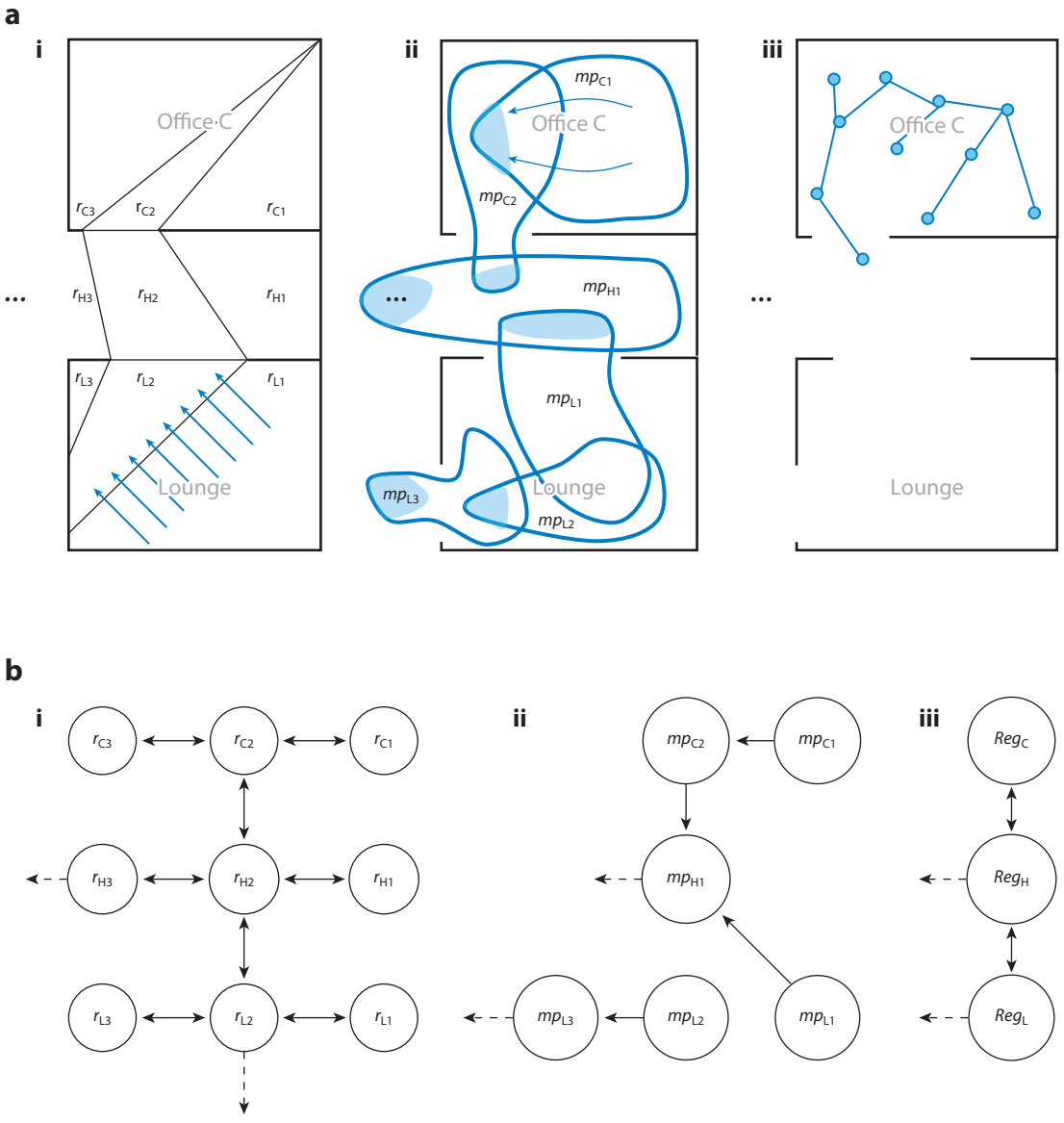

Figure 3

Abstractions and symbolic structures for the mail-delivery example. (*a*) Abstraction techniques. (*i*) Partition. The blue arrows represent the vector fields that would drive the robot from r_{L1} to r_{L2}. (*ii*) Motion primitives. The blue outlines of motion primitives mp_i represent their invariant sets r_i, and the blue regions represent their goal sets g_i. (*iii*) Motion planner. The blue graph represents the output of a motion planner that is searching for a path from office C to the hall. (*b*) Symbolic structures. (*i*) Partition. The arrows are bidirectional only if there exist controllers that can drive the robot between any adjacent regions [e.g., for holonomic robots (45–47)]. Depending on the partition algorithm, these arrows may become one-directional, and the symbolic structure may become nondeterministic. (*ii*) Motion primitives. An arrow exists between mp_i and mp_j if and only if $g_i \subseteq r_j$. (*iii*) Motion planner. The symbolic structure is initially fully connected (i.e., includes bidirectional arrows between adjacent regions). If the motion planner fails to find a path between regions i and j, then the arrow connecting Reg_i and Reg_j is removed.

For each motion primitive mp_i, r_i is the domain of the primitive and $g_i \subseteq r_i$ is the goal set that should be reached by activating mp_i. As before, overloading the notation, we use the symbol r_i as a label in the symbolic structure to mean that when a node is labeled with r_i, the physical system's state $x \in r_i \subseteq X$. Transitions in the discrete structure correspond to set inclusions of goals and domains of motion primitives; if $g_i \subseteq r_j$, then there exists a transition from r_i to r_j. If a transition exists, then motion primitive j can be activated after the completion of motion primitive i, but the reverse does not hold. **Figure 3** depicts a possible set of motion primitives for part of the workspace of the mail-delivery example (**Figure 3*a***, subpanel *ii*) and the corresponding symbolic structure (**Figure 3*b***, subpanel *ii*).

Note that with motion primitives, as opposed to partitions, the symbols are not mutually exclusive, nor do they always cover the workspace. If the specification is given over regions of the workspace, care must be taken to make sure those regions of space are fully covered by the motion primitives, and if no set of motion primitives can cover a region, the specification and region abstraction must be refined (30).

The notion of motion primitives is pervasive in robotics, and there are many different approaches to generating them, including reinforcement learning, learning by demonstration, and control theory. Enabling guarantees for the robot's physical behavior requires a simulation relation between the physical system and the abstraction. This means that the motion primitives that are suitable for synthesis must have two properties: invariance and liveness (or reachability). Invariance is the property that, when activating motion primitive mp_i, the state remains in the domain $x(t) \in r_i$; that is, the state will not exit the domain of the primitive. Liveness (or reachability) is the property that the state of the system will reach the goal set g_i in finite time from all states in the domain r_i. Approaches for generating motion primitives that satisfy the invariance and liveness properties include using the Hamilton–Jacobi formulation (56–58), creating vector fields over regions in the environment (46, 59), and sums-of-squares optimization for generating funnels around trajectories (60–64).

In the case of reactive synthesis, where the behavior of the robot might change owing to changes in the environment, to ensure correct execution with motion primitives, the abstraction must be reactively composable (30, 65). This means that there exists a set of motion primitives that enable the robot to "change its mind" and switch from one motion primitive to another before reaching its goal set. Moreover, designing abstractions for robots with multiple actuation capabilities requires special care with respect to timing semantics (66).

2.3.3. Motion planners and trajectories. In the previous two sections, the abstraction is created a priori based on the dynamics of the robot and the symbolic structure. Another approach is to start with a partition of the workspace and then, through iterative synthesis, search for robot trajectories that enable the selected transitions in the symbolic structure. This search can be done using motion planners such as sampling-based ones (67), optimization-based techniques such as model predictive control, and reachability computations similar to those discussed in Section 2.3.1.

Similar to the partition approach, the workspace is discretized into cells; however, in contrast to the partition approach, the cells do not have to observe dynamic constraints on the robot, and they typically correspond to regions that are semantically meaningful for the specifications. Formally, the workspace of the robot $\mathcal{W}$ is partitioned into regions Reg_i such that $Reg_i \cap Reg_j = \emptyset$ for all $i \neq j$ and $\cup_i Reg_i = \mathcal{W}$. Again overloading the notation, we use Reg_i as the symbol that is true when the robot is in region i of the partition. The transitions correspond to the adjacency of the regions; that is, if regions share a boundary, then there exists a bidirectional transition between these regions unless such a transition was removed during the iterative synthesis procedure. **Figure 3**

depicts a motion planner searching for a transition (**Figure 3*a***, subpanel *iii*) and the corresponding symbolic structure (**Figure 3*b***, subpanel *iii*).

This approach bypasses the need for detailed abstractions and is especially powerful for high-dimensional systems. The main consequence of this abstraction approach is relaxed or no guarantees. In open-loop synthesis, the synthesized controller is not guaranteed to be implementable on the physical system owing to a lack of simulation relation between the symbolic and physical systems—there might be a symbolic transition that is impossible to physically implement. In iterative synthesis, the guarantees are determined by the underlying motion planner. For instance, with sampling-based motion planners, probabilistic guarantees can be obtained.

3. SPECIFICATIONS

Synthesis is the process of transforming a specification (what the robot needs to do) into an implementation (how the robot will do it). Specifications can be roughly grouped into two types: safety specifications, which describe how the robot should always behave (e.g., "never collide with an obstacle" or "always maintain a line of sight from a base station") and liveness specifications, which describe goals or tasks/states that the robot must eventually achieve (e.g., "eventually go back to the recharge station" or "once in a while, send your location"). Safety and liveness specifications can be bounded over finite or infinite horizons, can be deterministic or probabilistic, and can be defined over different types of abstractions.

The majority of the work on synthesis for robots utilizes temporal logic to express desired robot behavior and assumptions about the behavior of the dynamic environment. Roughly speaking, temporal logic contains, in addition to Boolean operators, temporal operators that allow one to reason about the change in the truth value of propositions over time.

This section describes several specification formalisms that have emerged in recent years owing to developments in synthesis engines that make the synthesis process possible. To illustrate the expressive power of each formalism, Section 3.4 provides example specifications related to the mail-delivery example. The abstraction used is a partition of the workspace shown in **Figure 2**, where the nodes in the symbolic structure are the seven regions representing the rooms and the hallway.

3.1. Discrete Logics

There are different variants of discrete temporal logic (42); most of the work in synthesis for robots utilizes LTL, described below. It is worth noting that there are two notation conventions in the literature for the temporal operator: $\bigcirc$, $\Box$, $\diamond$, and $\mathcal{U}$, or X, G, F, and U. In this review, we follow the former convention.

3.1.1. Linear temporal logic syntax. Let AP be a set of atomic propositions where $\pi \in AP$ is a Boolean variable. LTL formulas are constructed from atomic propositions $\pi \in AP$ according to the following grammar:

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

where $\neg$ ("not") and $\vee$ ("or") are Boolean operators, and $\bigcirc$ ("next") and $\mathcal{U}$ ("until") are temporal operators. The Boolean constants true and false are defined as $True = \varphi \vee \neg\varphi$ and $False = \neg True$, respectively. Given negation ("not," $\neg$) and disjunction ("or," $\vee$), one can define conjunction ("and") $\varphi \wedge \varphi = \neg(\neg\varphi \vee \neg\varphi)$, implication ("if") $\varphi_1 \Rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$, and equivalence ("iff") $\varphi_1 \Leftrightarrow \varphi_2 = (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$.

Given the "next" ($\bigcirc$) and "until" ($\mathcal{U}$) temporal operators, additional temporal operators can be derived, such as "eventually" ($\diamond\varphi = True\,\mathcal{U}\,\varphi$) and "always" ($\square\varphi = \neg\diamond\neg\varphi$).

3.1.2. Linear temporal logic semantics. The semantics of an LTL formula φ are defined on an infinite sequence $\sigma = \sigma_1\sigma_2\ldots$ of truth assignments to the atomic propositions $\pi \in AP$, where σ_i denotes the set of atomic propositions that are true at position i. The recursive definition of whether σ satisfies LTL formula φ at position i (denoted $\sigma, i \models \varphi$) is as follows:

- $\sigma, i \models \pi$ iff $\pi \in \sigma_i$;
- $\sigma, i \models \neg\varphi$ iff $\sigma, i \not\models \varphi$;
- $\sigma, i \models \varphi_1 \vee \varphi_2$ iff $\sigma, i \models \varphi_1$, or $\sigma, i \models \varphi_2$;
- $\sigma, i \models \bigcirc\varphi$ iff $\sigma, i+1 \models \varphi$; and
- $\sigma, i \models \varphi_1\,\mathcal{U}\,\varphi_2$ iff there exists $k \geqslant i$ such that $\sigma, k \models \varphi_2$, and for all $i \leqslant j < k$, $\sigma, j \models \varphi_1$.

Intuitively, the formula $\bigcirc\varphi$ expresses that φ is true in the next step (the next position in the sequence), and 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 φ if $\sigma, 0 \models \varphi$. The sequence σ satisfies the formula $\square\varphi$ if φ is true in every position of the sequence, and it satisfies the formula $\diamond\varphi$ if φ is true at some position of the sequence.

3.1.3. Fragments of linear temporal logic. As described in Section 4, reactive synthesis for full LTL is computationally prohibitive (68). Therefore, researchers have explored several fragments of LTL that, while not as expressive as full LTL, are amenable to more tractable synthesis algorithms. Two fragments used by several researchers are the GR(1) (general reactivity of rank 1) fragment (69) and co-safe LTL (70).

3.1.3.1. The GR(1) fragment. Let the set $AP = \mathcal{X} \cup \mathcal{Y}$ be composed of $\mathcal{X}$, the set of propositions corresponding to the environment state as observed by sensors, and $\mathcal{Y}$, the set of propositions corresponding to the robot state, e.g., its position and actions.

LTL formulas in the GR(1) fragment (69) are of the form $\varphi = (\varphi_e \Rightarrow \varphi_s)$. The subformula φ_e is an assumption about the sensor propositions and thus about the behavior of the environment. An environment is considered admissible if it always satisfies the assumptions made about it in φ_e. Note that one does not have to make any assumptions about the environment; specifying $\varphi_e = True$ means that no assumptions are made. The formula φ_s represents the desired behavior of the robot.

The formula φ is true if φ_s is true (i.e., the desired robot behavior is satisfied) or φ_e is false (i.e., the environment did not behave as expected). This means that when the environment does not satisfy φ_e and is thus not admissible, there is no guarantee for the robot behavior.

Both φ_e and φ_s have the structure

$$\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e, \qquad \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s,$$

with the following:

- φ_i^e and φ_i^s are nontemporal Boolean formulas constraining (if at all) the initial value(s) of the sensor propositions $\mathcal{X}$ and robot propositions $\mathcal{Y}$, respectively.
- φ_t^e and φ_t^s represent safety assumptions and requirements (i.e., constraints that must always hold) for the environment and robot, respectively. For example, the assumption that a package will never be sensed when the robot is in the lounge belongs to φ_t^e, and motion constraints (e.g., if the robot is in office A, then in the next state it can be only in either office

A or the hallway) and behavior requirements (e.g., if the robot is carrying a package, then it may not go into the lounge) belong to φ_t^s.

- φ_g^e and φ_g^s represent liveness assumptions and requirements that must become true sometimes (or eventually) for the environment and robot, respectively. For example, an assumption that a package will eventually arrive can be part of φ_g^e, and a requirement that the robot eventually go to the lounge can be part of φ_g^s.

3.1.3.2. Co-safe linear temporal logic. This fragment includes LTL formulas whose truth value can be determined based on a finite sequence of truth assignments (70). The syntax is defined as

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

Note that negation is allowed only on propositions, not on formulas. This means that the operator "always" ($\Box$) is not part of the fragment (the truth value of a formula with $\Box$ can be evaluated only over infinite traces of a system), whereas "eventually" ($\Diamond$) is part of the fragment.

3.2. Probabilistic Logics

In robotics, it is natural to consider specifications that have a probabilistic nature. Rather than a deterministic set of requirements from the robot, the specification can include probabilities attached to the different task components. Probabilistic computation tree logic (PCTL) (adapted from 43) can capture these desired probabilities.

3.2.1. Probabilistic computation tree logic syntax. As with LTL, formulas are defined over a set of atomic propositions AP. PCTL formulas are state formulas defined recursively as follows:

$$\begin{aligned} &\text{state formulas:} && \varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathcal{P}_{\bowtie p}[\psi], \\ &\text{path formulas:} && \psi ::= \bigcirc\varphi \mid \varphi \,\mathcal{U}^{\leqslant k}\, \varphi \mid \varphi \,\mathcal{U}\, \varphi, \end{aligned}$$

where $\mathcal{P}$ is the probabilistic operator, $\bowtie \in \{<, \leqslant, \geqslant, >\}$, $p \in [0, 1]$, and $k \in \mathbb{N}$. State formulas φ are evaluated over the states of an MDP, while the path formulas ψ are assessed over paths and are allowed only as the parameter of the $\mathcal{P}$ operator.

As in LTL, conjunction ("and," $\wedge$), implication ("if," $\Rightarrow$), and equivalence ("iff," $\Leftrightarrow$) can be derived from negation ("not," $\neg$) and disjunction ("or," $\vee$), and "eventually" ($\Diamond$) can be derived from "until" ($\mathcal{U}$). By using $\Diamond$, the "always" ($\Box$) operator can be defined as $\mathcal{P}_{\bowtie p}[\Box\varphi] \equiv \mathcal{P}_{\bar{\bowtie} p}[\Diamond\neg\varphi]$, where $\bar{\leqslant} \equiv \geqslant$, $\bar{<} \equiv >$, $\bar{>} \equiv <$, and $\bar{\geqslant} \equiv \leqslant$ (71). Similarly, the bounded operators $\Diamond^{\leqslant k}$ and $\Box^{\leqslant k}$ can be defined using $\mathcal{U}^{\leqslant k}$.

3.2.2. Probabilistic computation tree logic semantics. PCTL formulas can be evaluated over either discrete-time Markov chains or MDPs. In this review, the underlying system model for probabilistic systems in an MDP $\mathcal{M} = (S, s_0, Act, Steps, AP, L)$; therefore, the semantics are introduced over MDPs. A state formula φ is satisfied in state $s \in S$ under policy μ as follows:

- $s \models \pi$ iff $\pi \in L(s)$;
- $s \models \neg\varphi$ iff $s \not\models \varphi$;
- $s \models \varphi_1 \vee \varphi_2$ iff $s \models \varphi_1$ or $s \models \varphi_2$; and
- $s \models \mathcal{P}_{\bowtie p}[\psi]$ iff $p_s^{\mu}(\psi) \bowtie p$,

where $p_s^{\mu}(\psi)$ is the probability of all (infinite) paths that satisfy ψ starting at state s under control policy μ.

A path formula ψ is satisfied over path $\omega \in Path_s$ as follows:

- $\omega \models \bigcirc\varphi$ iff $\omega_1 \models \varphi$;

- $\omega \models \varphi_1 \mathcal{U}^{\leqslant k} \varphi_2$ iff there exists $i \leqslant k$ such that $\omega_i \models \varphi_2$ and for all $j < i$, $\omega_j \models \varphi_1$; and
- $\omega \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists $k \geqslant 0$ such that $\omega \models \varphi_1 \mathcal{U}^{\leqslant k} \varphi_2$.

3.3. Metric Logics

The logics described above are defined over propositions that are Boolean variables that can be either true or false. Other logics, such as signal temporal logic (STL) (72), enable a richer specification language by allowing discrete-time continuous signals x_i and predicates over them to define the building blocks of the language.

3.3.1. Signal temporal logic syntax. Let AP^γ be a set of atomic predicates where $\pi^\gamma \in AP^\gamma$ is a predicate $X \to \{0, 1\}$ whose truth value corresponds to the sign of the function $\gamma : \mathbb{R}^{n_c} \to \mathbb{R}$. STL formulas are constructed from atomic predicates π^γ according to the following grammar:

$$\varphi ::= \pi^\gamma \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \mathcal{U}_{\langle a,b \rangle} \varphi,$$

where $\langle \in \{[, (\}, \rangle \in \{],)\}$ and $a, b \in \mathbb{R}_{\geqslant 0}$. As for LTL, conjunction ($\wedge$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$) can be derived from negation ($\neg$) and disjunction ($\vee$), and timed "eventually" ($\Diamond_{\langle a,b \rangle}$) and timed "always" ($\Box_{\langle a,b \rangle}$) can be derived from timed "until" ($\mathcal{U}_{\langle a,b \rangle}$). The main differences with respect to LTL are the notion of a predicate that replaces propositions (i.e., the continuous signal is explicitly abstracted through the function γ) and the notion of continuous intervals of time [which also renders the notion of "next" ($\bigcirc$) meaningless].

3.3.2. Signal temporal logic semantics. The satisfaction of an STL formula φ at time t is defined as follows:

- $(x, t_k) \models \pi^\gamma$ iff $\gamma(x(t_k)) > 0$;
- $(x, t_k) \models \neg\varphi$ iff $(x, t_k) \not\models \varphi$;
- $(x, t_k) \models \varphi_1 \vee \varphi_2$ iff $(x, t_k) \models \varphi_1$ or $(x, t_k) \models \varphi_2$; and
- $(x, t_k) \models \varphi_1 \mathcal{U}_{\langle a,b \rangle} \varphi_2$ iff there exists $t_{k'} \in \langle t_k + a, t_k + b \rangle$ such that $(x, t_{k'}) \models \varphi_2$, and for all $t_{k''} \in [t_k, t_{k'}]$, $(x, t_{k''}) \models \varphi_1$.

A projection of ξ onto the state space, $\mathbf{x} = x_0 x_1 x_2 \ldots$, satisfies φ, denoted by $\mathbf{x} \models \varphi$, if $(\mathbf{x}, t_0) \models \varphi$. Informally, $(a)\, \mathbf{x} \models \Box_{[a,b]}\varphi$ if φ holds for all time between a and b and $(b)\, \mathbf{x} \models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ if φ_1 holds at every time step before φ_2 holds and if φ_2 holds at some time step between a and b. Additionally, $\Diamond_{[a,b]}\varphi = True \mathcal{U}_{[a,b]}\varphi$, which is true if φ holds at some time step between a and b.

An STL formula φ is bounded-time if it contains no unbounded operators; the bound of φ is the maximum over the sums of all nested upper bounds on the temporal operators and provides a conservative maximum trajectory length required to decide its satisfiability. For example, for $\Box_{[0,10]} \Diamond_{[1,6]}\varphi$, a trajectory of length $N \geq 10 + 6 = 16$ is sufficient to determine whether the formula is true.

A unique property of STL is that the formalism admits a quantitative semantics that, in addition to the yes/no answer to the satisfaction question, provides a real number $\rho^\varphi(x, t)$, called the robustness of satisfaction, that grades the quality of the satisfaction or violation (73). The robustness score $\rho^\varphi(x, t)$ is computed recursively on the structure of the formula just like the Boolean semantics and is defined such that $(x, t) \models \varphi \iff \rho^\varphi(x, t) > 0$. The robustness score should be interpreted as how much the model satisfies φ; its absolute value corresponds to the distance of x from the set of trajectories satisfying or violating φ. Such semantics have also been defined for other timed logics, including metric temporal logic (MTL) (74), to assess the robustness of the systems to parameter or timing variations.

3.4. Example Specifications for the Mail-Delivery Scenario

Given the abstraction, the set of propositions $AP = \{Office_A, Office_B, Office_C, Office_D, Lounge, Mailroom, Hallway, Pickup, Deliver, SensePackage\}$ includes the regions, the pick-up and deliver actions, and the sensor that detects when a package is available. The set of functions γ_{Regi} return true when the position of the robot is in region i. The following are example encodings of specifications in the different logics:

- Co-safe LTL (nonreactive): $(\neg Deliver)\, \mathcal{U}(Deliver \wedge (Office_B \vee Office_C))$ expresses "deliver a package to one of offices B or C but not anywhere else."
- GR(1) fragment (reactive): $\square\diamond(SensePackage \rightarrow (Deliver \wedge (Office_B \vee Office_C))) \wedge \square\diamond(\neg SensePackage \rightarrow Mailroom)$ expresses "if you sense a package, then deliver it to office B or office C; otherwise, go to the mail room" (only part of the formula is shown).
- PCTL: $\mathcal{P}_{>0.95}(\diamond(Deliver \wedge (Office_B \vee Office_C)))$ expresses "the robot should deliver the package to office B or office C with a probability greater than 0.95."
- STL: $\diamond_{[0,5]}(Deliver \wedge (Office_B \vee Office_C))$ expresses "deliver the package to office B or office C within 5 time units."

4. SYNTHESIS ALGORITHMS

Having described different abstractions and specification formalisms in the previous sections, we now provide an overview of the synthesis algorithms used to synthesize controllers. **Table 1** summarizes the abstraction, specification formalism, and structure of the resulting controller for each algorithm.

4.1. Automata-Based Synthesis Algorithms

Automata-based synthesis algorithms are defined for systems that are abstracted as a Kripke structure $\mathcal{K}$. The specifications are given as LTL formulas, and the abstraction $\mathcal{K}$ is deterministic; that is, there is a unique initial state $S_0 = \{s_0\}$, and every transition $(s, s') \in R$ can be chosen by the robot. The algorithms for probabilistic and nondeterministic systems are discussed in Sections 4.2 and 4.3.

Automata-based synthesis methods generally include three main steps: translation of the specification into an automaton, composition of the system abstraction with the automaton, and

Table 1 Abstractions, specifications, and synthesis products for different algorithms

Algorithm	Abstraction	Specification	Synthesis product
Automata-based algorithms	Deterministic Kripke structure	LTL	Sequence of states
PCTL for MDPs	MDP	PCTL	Policy mapping finite paths to an action
LTL for MDPs	MDP	LTL	Policy mapping finite paths to an action
Game-based algorithms	Robot and environment transitions are part of the specification	LTL, GR(1) fragment of LTL	Finite state controller
Optimization-based algorithms	Difference equation, polytopic regions of interest	LTL, MTL, or STL, and a cost function J	Control sequence

Abbreviations: LTL, linear temporal logic; MDP, Markov decision process; MTL, metric temporal logic; PCTL, probabilistic computation tree logic; STL, signal temporal logic.

computation of an accepting path or policy over the composed system. For example, an LTL formula can be automatically transformed into a nondeterministic Büchi automaton (NBA) that accepts precisely those traces that satisfy the formula (42, 71). Given an abstraction $\mathcal{K}$ over AP and an LTL formula φ also defined over AP, the automata-based algorithms generate the product structure $\mathcal{K}^{\mathcal{P}} = \mathcal{K} \times \mathcal{A}$, where $\mathcal{A}$ is an NBA constructed from φ with a set of state Z, a set of input alphabets $\Sigma = 2^{AP}$, a transition function $\delta : Z \times \Sigma \to 2^{Z}$, and a set of accepting states $F^{\mathrm{NBA}} \subseteq Z$. In $\mathcal{K}^{\mathcal{P}}$, the set of states is $S \times Z$, and a transition exists from state (s, z) to (s', z') if $(s, s') \in R$ and $z' \in \delta(z, L(s'))$, where $s, s' \in S$ and $z, z' \in Z$. Let $\omega^{\mathcal{P}}$ denote an infinite path of $\mathcal{K}^{\mathcal{P}}$ that visits the states in $F^{\mathcal{P}} = S \times F^{\mathrm{NBA}}$ infinitely often. The projection of $\omega^{\mathcal{P}}$ onto $\mathcal{A}$ is an accepting run satisfying φ, and the projection of $\omega^{\mathcal{P}}$ onto $\mathcal{K}$ is an abstraction path that also satisfies φ. Therefore, the synthesis problem is reduced to finding $\omega^{\mathcal{P}}$ over $\mathcal{K}^{\mathcal{P}}$. This computation boils down to two graph search steps: (*a*) identifying the cycles in $\mathcal{K}^{\mathcal{P}}$ that contain at least one accepting state in $F^{\mathcal{P}}$ and (*b*) finding a path from an initial state to one of the cycles. This results in a path $\omega^{\mathcal{P}} = \underline{\omega}^{\mathcal{P}} \overline{\omega}^{\mathcal{P}}$, where $\underline{\omega}^{\mathcal{P}}$ is a finite path, known as the prefix, and $\overline{\omega}^{\mathcal{P}}$ is an infinite repeat of a cycle, known as the suffix.

Similarly, from a co-safe LTL formula φ, a deterministic finite automaton (DFA) can be constructed that accepts only the finite traces that satisfy φ (70). For DFAs, in the synthesis algorithm, after obtaining the product $\mathcal{K}^{\mathcal{P}} = \mathcal{K} \times \mathcal{A}$, it is enough to find a finite path (i.e., $\omega^{\mathcal{P}} = \underline{\omega}^{\mathcal{P}}$) to an accepting state in $F^{\mathcal{P}} = S \times F^{\mathrm{DFA}}$ from the initial state. The computational bottleneck for automata-based synthesis algorithms lies in the translation of the formula to the automaton. The complexity of this translation is exponential in the size of the formula for LTL to NBA and doubly exponential for co-safe LTL to DFA.

Most work on automata-based synthesis for robot control follows the general steps of the algorithm described above (e.g., 75–82). The differences typically lie in the abstraction step (underlying dynamical system) or the generation of $\omega^{\mathcal{P}}$ with a desired property. For example, Reference 76 introduced an end-to-end LTL synthesis framework for linear dynamical systems through an automated construction of a bisimilar (equivalent) abstraction using a simplex-based discretization. Techniques for path optimization have been studied, where the costs are typically defined over the transitions of the abstraction (79). In some work, to avoid the complexity of abstraction (for nonlinear systems), a coarse abstraction and the specification automaton are used to guide the search for a feasible trajectory by reducing the problem to a series of constrained reachability problems (80). In Reference 75, instead of explicitly constructing the specification automaton, model checkers were used to find a path that satisfies the specification.

To deal with complex and/or high-dimensional dynamical systems, sampling-based motion planning has been introduced (e.g., 81–87). With these techniques, however, it is difficult (if not impossible) to obtain a cyclic behavior for the robot. Therefore, these frameworks focus on co-safe LTL formulas, which allow the expression of tasks that can be achieved in finite time. Furthermore, these approaches typically consist of layers of planners. At the highest level, the DFA $\mathcal{A}$ and the abstraction $\mathcal{K}$ are employed to guide (suggest finite paths for) the exploration of the state space for feasible solutions by the low-level sampling-based planner. Depending on whether the low-level motion planner found a path, the feasibility of the transitions in $\mathcal{K}$ are learned during the planning procedure, leading to ever-improving high-level plans (guidance).

4.2. Markov Decision Process–Based Synthesis Algorithms

MDP-based synthesis focuses on generating a policy that maximizes (or in some cases minimizes) the probability of satisfying the specification. The first work in MDP-based synthesis (88–91) focused on specifications given as probabilistic logic formulas, namely PCTL (43). These

Table 2 Probability optimization formulations for Markov decision process–based synthesis

PCTL formula	Optimization formulation
$\mathcal{P}_{\max=?}[\bigcirc\varphi_1]$	$\mathfrak{p}_s^* = \max_{a\in A(s)} \sum_{s'\in \mathrm{Sat}(\varphi_1)} \mathit{Steps}(s,a)(s')$
$\mathcal{P}_{\max=?}[\varphi_1 \mathcal{U}^{\leq k} \varphi_2]$	$\mathfrak{p}_s^k = \max_{a\in A(s)}(\sum_{s'\in S^?} \mathit{Steps}(s,a)(s') \cdot \mathfrak{p}_{s'}^{k-1} + \sum_{s'\in S^{\mathrm{yes}}} \mathit{Steps}(s,a)(s'))$
$\mathcal{P}_{\max=?}[\varphi_1 \mathcal{U} \varphi_2]$	$\min \sum_{s\in S^?} \mathfrak{p}_s \quad \text{subject to} \quad \mathfrak{p}_s \geqslant \sum_{s'\in S^?} \mathit{Steps}(s,a)(s') \cdot \mathfrak{p}_{s'} + \sum_{s'\in S^{\mathrm{yes}}} \mathit{Steps}(s,a)(s')$

Abbreviation: PCTL, probabilistic computation tree logic.

specifications are natural for probabilistic systems, and their synthesis algorithms have polynomial complexity. Nevertheless, the syntax of PCTL is constrained to one temporal operator per path formula, whereas in LTL, temporal operators can be combined and nested to specify complex tasks. The major challenge in LTL synthesis for probabilistic (noisy) systems is dealing with infinite runs, which usually result in a zero probability of satisfaction if a transition probability between two recurring states in the path is less than 1. Therefore, the initial work in LTL synthesis for MDPs focused on low-level controllers to reduce the stochastic nature of the system (92). By exploiting the end components of MDPs, later work introduced full LTL synthesis algorithms for MDPs (93–96), which involve solving an optimization problem over a structure whose size is doubly exponential in the length of the LTL formula. To overcome the computational burden for high-dimensional systems in large environments, the use of learning algorithms has been explored in both PCTL and LTL synthesis (97–101). In recent years, synthesis methods for uncertain MDPs have been studied to relax the single-valued transition probability of MDPs (102–104). By allowing uncertainty over the transition probabilities, these models arguably provide a better modeling framework for physical systems with noise than classical MDPs because it is typically difficult to compute an exact transition probability for such systems.

4.2.1. Probabilistic computation tree logic synthesis for Markov decision processes. The PCTL control synthesis algorithm for MDPs takes a PCTL formula φ and an MDP $\mathcal{M}$ and returns both the optimal probability of satisfying φ and the corresponding control policy (88, 89, 91). The basic algorithm proceeds by constructing the parse tree for φ and treating each operator in the formula separately.

For the formula $\mathcal{P}_{\max=?}[\bigcirc\varphi_1]$, the objective is to determine the action that produces the maximum probability of satisfying $\bigcirc\varphi_1$ at each MDP state. Thus, only the immediate transitions at each state need to be considered, which reduces the optimization problem to the one shown in **Table 2**, where $\mathfrak{p}_s^*$ denotes the optimal probability of satisfying φ at the state $s \in S$, and $\mathrm{Sat}(\varphi_1) \subseteq S$ is the set of states that satisfy φ_1. This optimization problem can be solved by a matrix-vector multiplication (91).

For formulas $\mathcal{P}_{\max=?}[\varphi_1 \mathcal{U}^{\leq k} \varphi_2]$ and $\mathcal{P}_{\max=?}[\varphi_1 \mathcal{U} \varphi_2]$, first the MDP states are grouped into three subsets: states that always satisfy the specification S^{yes}, states that never satisfy the specification S^{no}, and the remaining states $S^?$. Trivially, the probabilities of the states in S^{yes} and in S^{no} are 1 and 0, respectively. For $\mathcal{U}^{\leq k}$, the probabilities of the remaining states $s \in S^?$ are defined recursively as shown in **Table 2**, which can be computed by k matrix-vector multiplications. This results in a time-dependent policy; that is, for each time index k, an action is assigned to each satisfying state. For $\mathcal{U}$, the computation for the states in $S^?$ is known as the maximal reachability probability problem (MRPP) (105), which can be solved by the linear programming problem shown in **Table 2**. The complexity of this method is polynomial in the size of the MDP $\mathcal{M}$, which is $|\mathcal{M}| = \sum_{s\in S} |A(s)|$, and the obtained control policy is stationary. The MRPP can also be solved by using value iteration,

which is essentially solving for $\mathcal{U}^{\leq k}$ with the termination rule of convergence of the probability values, i.e., $p_s^{\max}(\varphi_1 \mathcal{U}^{\leq k} \varphi_2) \approx p_s^{\max}(\varphi_1 \mathcal{U}^{\leq k-1} \varphi_2)$ for all $s \in S$.

4.2.2. Linear temporal logic synthesis for Markov decision processes. The LTL control synthesis algorithm for MDPs takes an LTL formula φ and an MDP $\mathcal{M}$ and returns both the optimal probability of satisfying φ and the corresponding control policy (93, 95). The algorithm follows the general method of automata-based synthesis (Section 4.1). That is, first an automaton is constructed from φ, and then a control policy is computed on the product of the automaton with the MDP.

In LTL synthesis for MDPs, instead of an NBA, a deterministic Rabin automaton (71) is generated from the LTL formula φ. The product $\mathcal{M}^{\mathcal{P}} = \mathcal{M} \times \mathcal{A}$ is then generated similarly to the deterministic case (Section 4.1). Product $\mathcal{M}^{\mathcal{P}}$ is an MDP whose transition probability function $Steps^{\mathcal{P}}$ is defined in accordance to $Steps$ as follows: $Steps^{\mathcal{P}}((s,z),a^{\mathcal{P}})((s',z')) = Steps(s,a)(s')$ if $z' = \delta(z, L(s'))$; 0 otherwise. Next, a graph search is performed on $\mathcal{M}^{\mathcal{P}}$ to identify the end components that enable the satisfaction of the deterministic Rabin automaton–accepting condition, which requires infinite visits of some accepting states, while some particular states need to be visited finitely often. The synthesis problem is then reduced to the computation of the optimal policy that maximizes the probability of reaching these accepting end components over $\mathcal{M}^{\mathcal{P}}$. This problem is equivalent to solving the MRPP, whose linear programming formulation in shown **Table 2**. For the case that φ is co-safe LTL, the MRPP is set up to maximize the probability of reaching the $\mathcal{M}^{\mathcal{P}}$ states that correspond to the accepting states of the DFA.

For both co-safe and full LTL formulas, the obtained optimal policy is stationary (history independent) on $\mathcal{M}^{\mathcal{P}}$. The policy can be mapped to the states and actions of $\mathcal{M}$, in which case it becomes history dependent. Therefore, during the execution of the policy by the robot, it is necessary to keep track of the robot's evolution over the states of $\mathcal{M}^{\mathcal{P}}$. The complexity of this LTL synthesis algorithm for MDPs is polynomial in the size of the product MDP, which itself is doubly exponential in the size of the LTL formula φ.

4.3. Game-Based Synthesis Algorithms

Recall from Section 3 that in reactive synthesis of robot controllers, the set of propositions AP is divided into two sets: sensor propositions ($\mathcal{X}$) and robot propositions ($\mathcal{Y}$). An LTL formula φ is realizable if there exists a finite state strategy that, for every finite sequence of truth assignments to the sensor propositions, provides an assignment to the robot propositions such that every infinite sequence of truth assignments to both sets of propositions generated in this manner satisfies φ. The synthesis problem is to find a finite state controller (if one exists) that encodes this strategy, i.e., whose executions correspond to sequences of truth assignments that satisfy φ. Synthesis of reactive systems has high computational complexity for many specification languages. For an arbitrary LTL formula, the complexity of the synthesis algorithm is doubly exponential in the size of the formula (68). However, when restricted to formulas of the GR(1) fragment, the algorithm in Reference 69 permits synthesis in time polynomial in the size of the abstracted state space. The question of realizability is viewed as a two-player game between the robot and the environment, who have to play according to the transition rules defined by φ_e^i, φ_e^t, φ_s^i, and φ_e^t. The winning condition—referred to as the GR(1) condition—is provided by $\varphi_e^g \Rightarrow \varphi_s^g$. The utility of specifications of this form has been demonstrated in a variety of robotic contexts (e.g., 106–108).

Synthesis from GR(1) specifications reduces to solving a μ-calculus fixed-point equation with three nested fixed points on a game structure that is built from the specification. The transitions in the game structure are given by the safety assumptions and guarantees (in contrast to

automata-based approaches, wherein the transition system is usually provided separately), and the liveness properties are translated to environment and robot goals, which the robot and environment players try to satisfy infinitely often.

The semantics of μ-calculus formulas can be found in References 69 and 109; an informal summary of the relevant portions is as follows:

- Q is the set of game states, and $[\![\varphi]\!]$ is the set of states that satisfy φ.
- $[\![\textcircled{\scriptsize}\varphi]\!]$ is the set of states $Q' \subseteq Q$ from which the robot can enforce that the next state will be in $[\![\varphi]\!]$, regardless of what the environment does next (i.e., for every $x \in 2^X$).
- $[\![\mu Q.\psi(Q)]\!]$ is a least fixed-point operation, computing the smallest set of states Q satisfying $Q = \psi(Q)$.
- $[\![\nu Q.\psi(Q)]\!]$ is a greatest fixed-point operation, computing the largest set of states Q satisfying $Q = \psi(Q)$.

In Reference 69, the set of winning states for the robot is characterized by the μ-calculus formula

$$\varphi_{\text{win}} = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_n \end{bmatrix} . \begin{bmatrix} \mu Y. \left(\bigvee_{i=1}^{m} \nu X. (J_s^1 \wedge \textcircled{\scriptsize} Z_2 \vee \textcircled{\scriptsize} Y \vee \neg J_e^i \wedge \textcircled{\scriptsize} X) \right) \\ \mu Y. \left(\bigvee_{i=1}^{m} \nu X. (J_s^2 \wedge \textcircled{\scriptsize} Z_3 \vee \textcircled{\scriptsize} Y \vee \neg J_e^i \wedge \textcircled{\scriptsize} X) \right) \\ \vdots \\ \mu Y. \left(\bigvee_{i=1}^{m} \nu X. (J_s^n \wedge \textcircled{\scriptsize} Z_1 \vee \textcircled{\scriptsize} Y \vee \neg J_e^i \wedge \textcircled{\scriptsize} X) \right) \end{bmatrix}, \quad 1.$$

where J_e^i is the ith environment liveness from φ_e^g ($i \in \{1, \ldots, m\}$), and J_s^j is the jth robot liveness from φ_s^g ($j \in \{1, \ldots, n\}$). Let $\oplus$ denote summation modulo n. For $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$, the greatest fixed point $\nu X.(J_s^j \wedge \textcircled{\scriptsize} Z_{j\oplus 1} \vee \textcircled{\scriptsize} Y \vee \neg J_e^i \wedge \textcircled{\scriptsize} X)$ characterizes the set of states from which the robot can force the game to stay infinitely in states satisfying $\neg J_e^i$, thus falsifying the left-hand side of the implication $\varphi_e \Rightarrow \varphi_s$, or in a finite number of steps reach a state in the set $Q_{\text{win}} = [\![J_s^j \wedge \textcircled{\scriptsize} Z_{j\oplus 1} \vee \textcircled{\scriptsize} Y]\!]$. The two outer fixed points ensure that the robot wins from the set Q_{win}: μY ensures that the play reaches a $J_s^j \wedge \textcircled{\scriptsize} Z_{j\oplus 1}$ state in a finite number of steps, and νZ ensures that the robot can loop through the livenesses in cyclic order. From the intermediate steps of the above computation, a state machine that realizes the specification is extracted, provided that every initial state is winning (69).

The GR(1) synthesis problem corresponds to solving Equation 1 and has complexity quadratic in the size of the game structure, i.e., $O(|Q|^2)$, which is still exponential in the number of atomic propositions in the specification. This synthesis algorithm has been extended to efficiently accommodate several fragments of LTL (66, 110, 111). Moreover, open source tools such as JTLV (Java Temporal Logic Verifier) (112) and Slugs (Small but Complete GR One Synthesizer) (111) do not build the game structure explicitly but use binary decision diagrams as a symbolic data structure to efficiently solve Equation 1. Robotics and control-specific tools such as LTLMoP (Linear Temporal Logic Mission Planning) (113) and TuLiP (Temporal Logic Planner) (114) have leveraged these implementations to provide domain-specific interfaces for operating in the real world. These tools take care of all phases of synthesis other than the discrete logical synthesis, i.e., specification (via a graphical user interface), abstraction (through the use of robot-specific controllers), and execution (in simulation or on a physical platform).

4.4. Optimization-Based Synthesis Algorithms

Optimization-based approaches consider difference equations (Section 2.1) and take as input an LTL, MTL, or STL formula φ; a cost function of the form $J(x_0, u, w, \varphi) \in \mathbb{R}$; an initial state $x_0 \in X$;

Table 3 Optimization-based synthesis formulations

	Nonreactive	Receding horizon	Reactive
Find	$\underset{\mathbf{u}\in\mathcal{U}^N}{\operatorname{argmin}} J(x_0, \mathbf{u}, \mathbf{w}, \varphi)$	$\underset{\mathbf{u}_K^L\in\mathcal{U}^L}{\operatorname{argmin}} J(x_k, \mathbf{u}_k^L, \mathbf{w}_k, \varphi)$	$\underset{\mathbf{u}^N\in\mathcal{U}^N}{\operatorname{argmin}} \max_{\mathbf{w}^N\in\mathcal{W}^N, \mathbf{w}\models\varphi_e} J(\xi(x_0, \mathbf{u}^N, \mathbf{w}^N))$
s.t.	$\xi(x_0, \mathbf{u}, \mathbf{w}) \models \varphi$	$\xi(x_0, \mathbf{u}, \mathbf{w}) \models \varphi$	$\forall \mathbf{w}^N \in \mathcal{W}^N, \xi(x_0, \mathbf{u}^N, \mathbf{w}^N) \models \varphi$

a horizon L; and, optionally, an external environmental input signal of length N, $\mathbf{w} \in \mathcal{W}^N$. The three types of problems described in Section 1 are encoded and solved as follows (see the formal summary in **Table 3**):

- Open loop (nonreactive): Find a control signal $\mathbf{u}$ of length N given a nominal environment $\mathbf{w}$ (73, 115–118). The state of the robot is assumed to be fully observable, and the environment inputs are known in advance. To allow the interpretation of specifications over infinite sequences of states, a prefix–suffix trajectory parameterization is usually used.
- Iterative: Find a control signal $\mathbf{u}$ over a finite horizon L assuming that the environment can change at each iteration k, but there exists a reliable prediction of it $\mathbf{w}_k$ over the horizon L. Such receding-horizon or model predictive control problems are solved iteratively online: At each time step, only the first control input in the sequence is implemented, and the problem is solved again (73, 119, 120). In References 73 and 120, a control input was synthesized for infinite sequences satisfying $\varphi = G\psi$ for formulas ψ with bound H by repeatedly synthesizing control for sequences of length $L = 2H$.
- Reactive: Find a control signal $\mathbf{u}$ of length N given a possibly adversarial, a priori uncertain environment $\mathbf{w}$. As in the game-based synthesis approach (Section 4.3), the environment is assumed to satisfy the temporal logic formula φ_e (120), and the controllers produced provide guarantees for specifications of the form $\varphi_e \Rightarrow \varphi_s$. For specification logics such as STL that admit quantitative semantics, this problem is solved as a two-player game, where the environment tries to minimize the quantitative satisfaction of the specification, while the robot simultaneously tries to maximize it. In Reference 120, counterexamples are used to inductively refine the synthesized controller until convergence or a maximum number of iterations is reached.

Additional treatments for systems with uncertainty have also been proposed (121, 122) but are beyond the scope of this review.

All of the above problem formulations include constraints on system evolution, based on the modeled dynamics, and desired robot behavior encoded as temporal logic formulas (which may be reactive, i.e., implications). Temporal logic constraints are encoded in various ways, exploiting properties of the underlying logic. For example, Reference 73 showed how the robustness of an STL specification φ can be recursively encoded using mixed integer–linear program (MILP) constraints, and enforcing $\rho^{\varphi}(x, t) > 0$ ensures the satisfaction of the formula. In Reference 117, reach–avoid-type LTL specifications over regions of interest that correspond to unions of convex polytopes are encoded as MILP as well. Other approaches (115, 116, 122) are tailored to specific fragments of temporal logic.

In all of the above approaches, the union of temporal logic constraints and robot constraints yields a single mathematical program (which is an MILP for linear or piecewise-linear robot dynamics), which can be checked for feasibility and solved when possible using an off-the-shelf (MILP) solver. Given an objective function on runs of the system, it is also possible to find an optimal trajectory that satisfies the logical specification. The robustness provides a natural objective, either in the absence of or as a complement to domain-specific objectives on runs of the system.

MILPs are NP hard, but the computational costs of an MILP encoding can be described in terms of the number of variables and constraints. In optimization-based approaches to synthesis, if N is the length of the desired control signal, P is the set of predicates used in the formula, and $|\varphi|$ is the length (i.e., the number of operators), then $O(N \cdot |P|) + O(N \cdot |\varphi|)$ continuous variables are introduced. In addition, $O(N)$ binary variables are introduced for every instance of a Boolean operator, i.e., $O(N \cdot |\varphi|)$ Boolean variables. The dimensionality of the discrete-time system also affects the size of the constructed MILP linearly via the constraints encoding system evolution (more precisely, through the size of the set of predicates P).

5. CONCLUSION

This article has described formal synthesis for robot control: the models used to define the problem; the specification formalisms used to capture complex, high-level tasks; and the main algorithms used to automatically transform the specifications into implementations that can be used to control physical robots. Synthesis is a powerful technique for increasing robot reliability because it provides guarantees with respect to the model and feedback regarding the specifications and model. Furthermore, by allowing a person to reason about the specification and not the implementation, synthesis reduces the time to deployment of a new task while eliminating human error in the implementation. Its main advantages are manifested when considering complex tasks with different constraints, reactions to events in the environment, and goals, and also when a robot needs to quickly change tasks and/or environment.

Synthesis for robotics is a growing field with a high potential impact, but the techniques are not yet widely used. Some robotic tasks of high interest in the community, such as going to a goal location in a cluttered workspace (for motion or manipulation), are better served using motion planners or learned controllers. Complex robotic systems, such as humanoid robots, are difficult to model and abstract—the size of the resulting symbolic structure is either too large to synthesize over or too small (meaning that the abstraction is too coarse) to enable a simulation relation with the continuous physical system. Existing software for the control and actuation of robots is not typically written in a way that can be easily abstracted into a symbolic model. Writing specifications instead of implementations requires both a paradigm shift in how robotic systems are deployed and expertise in the specification formalisms. As synthesis techniques mature and symbolic models are developed for different robot platforms, we expect synthesis to become an important tool in the robust and reliable deployment of robotic systems.

FUTURE ISSUES

1. Abstractions: The choice of abstraction level affects both the scalability of synthesis and the guarantees it can provide. Too fine grained an abstraction causes synthesis to becomes intractable, and too coarse an abstraction causes the models to lose fidelity with respect to the physical system, in which case synthesis can no longer provide realistic guarantees. Techniques for developing abstractions that are task, robot, and environment dependent are an active area of research.
2. Synthesis and learning: There is great potential in combining formal synthesis and learning to scale up synthesis, create abstractions that can be used for synthesis, and create explainable artificial intelligence. The challenge is maintaining the guarantees while leveraging data-driven approaches to control.

3. Synthesis and human–robot interaction: Formalizing models and creating abstractions and specifications for human–robot interaction will enable the synthesis of robot controllers and feedback that are task, environment, and interaction dependent, thus creating robots that can explain and guarantee their behavior in a human–robot interaction setting.

DISCLOSURE STATEMENT

The authors are not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

LITERATURE CITED

1. Clarke EM, Wing JM. 1996. Formal methods: state of the art and future directions. *ACM Comput. Surv.* 28:626–43
2. Spenko M, Buerger S, Iagnemma K, eds. 2017. The DARPA Robotics Challenge finals. *J. Field Robot.* 34(2, Spec. Issue). New York: Wiley
3. Loizou SSG, Kyriakopoulos KJ. 2004. Automatic synthesis of multiagent motion tasks based on LTL specifications. In *43rd IEEE Conference on Decision and Control*, pp. 153–58. New York: IEEE
4. Kloetzer M, Belta C. 2007. Temporal logic planning and control of robotic swarms by hierarchical abstractions. *IEEE Trans. Robot.* 23:320–30
5. Karaman S, Frazzoli E. 2008. Complex mission optimization for Multiple-UAVs using Linear Temporal Logic. In *2008 American Control Conference*, pp. 2003–9. New York: IEEE
6. Kress-Gazit H, Ayanian N, Pappas GJ, Kumar V. 2008. Recycling controllers. In *2008 IEEE International Conference on Automation Science and Engineering*, pp. 772–77. New York: IEEE
7. Filippidis I, Dimarogonas DV, Kyriakopoulos KJ. 2012. Decentralized multi-agent control from local LTL specifications. In *2012 IEEE 51st Annual Conference on Decision and Control (CDC)*, pp. 6235–40. New York: IEEE
8. Wongpiromsarn T, Ulusoy A, Belta C, Frazzoli E, Rus D. 2013. Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5011–18. New York: IEEE
9. Raman V, Kress-Gazit H. 2014. Synthesis for multi-robot controllers with interleaved motion. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4316–21. New York: IEEE
10. Raman V. 2014. Reactive switching protocols for multi-robot high-level tasks. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 336–41. New York: IEEE
11. DeCastro JA, Alonso-Mora J, Raman V, Rus D, Kress-Gazit H. 2018. Collision-free reactive mission and motion planning for multi-robot systems. In *Robotics Research*, Vol. 1, ed. A Bicchi, W Burgard, pp. 459–76. Springer Proc. Adv. Robot. Vol. 2. Cham, Switz.: Springer
12. Wong KW, Kress-Gazit H. 2015. Let's talk: autonomous conflict resolution for robots carrying out individual high-level tasks in a shared workspace. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 339–45. New York: IEEE
13. Tumova J, Dimarogonas DV. 2016. Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica* 70:239–48
14. Saha I, Ramaithitima R, Kumar V, Pappas GJ, Seshia SA. 2014. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1525–32. New York: IEEE
15. Nedunuri S, Prabhu S, Moll M, Chaudhuri S, Kavraki LE. 2014. SMT-based synthesis of integrated task and motion plans from plan outlines. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 655–62. New York: IEEE

16. Shoukry Y, Nuzzo P, Saha I, Sangiovanni-Vincentelli AL, Seshia SA, et al. 2016. Scalable lazy SMT-based motion planning. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 6683–88. New York: IEEE
17. Raman V, Kress-Gazit H. 2013. Explaining impossible high-level robot behaviors. *IEEE Trans. Robot.* 29:94–104
18. Lignos C, Raman V, Finucane C, Marcus M, Kress-Gazit H. 2014. Provably correct reactive control from natural language. *Auton. Robots* 38:89–105
19. DeCastro JA, Ehlers R, Rungger M, Balkan A, Kress-Gazit H. 2017. Automated generation of dynamics-based runtime certificates for high-level control. *Discrete Event Dyn. Syst. Theory Appl.* 27:371–405
20. Dokhanchi A, Hoxha B, Fainekos G. 2015. Metric interval temporal logic specification elicitation and debugging. In *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pp. 70–79. New York: IEEE
21. Fainekos GE. 2011. Revising temporal logic specifications for motion planning. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 40–45. New York: IEEE
22. Tumova J, Hall GC, Karaman S, Frazzoli E, Rus D. 2013. Least-violating control strategy synthesis with safety rules. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, pp. 1–10. New York: ACM
23. Guo M, Johansson KH, Dimarogonas DV. 2013. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5025–32. New York: IEEE
24. Kim K, Fainekos G, Sankaranarayanan S. 2015. On the minimal revision problem of specification automata. *Int. J. Robot. Res.* 34:1515–35
25. Lahijanian M, Almagor S, Fried D, Kavraki LE, Vardi MY. 2015. This time the robot settles for a cost: a quantitative approach to temporal logic planning with partial satisfaction. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 3664–71. Menlo Park, CA: AAAI Press
26. Lahijanian M, Kwiatkowska M. 2016. Specification revision for Markov decision processes with optimal trade-off. In *2016 IEEE 55th Conference on Decision and Control*, pp. 7411–18. New York: IEEE
27. Li W, Dworkin L, Seshia SA. 2011. Mining assumptions for synthesis. In *2011 9th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pp. 43–50. New York: IEEE
28. Alur R, Moarref S, Topcu U. 2013. Counter-strategy guided refinement of GR(1) temporal logic specifications. In *2013 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 26–33. New York: IEEE
29. DeCastro JA, Raman V, Kress-Gazit H. 2015. Dynamics-driven adaptive abstraction for reactive high-level mission and motion planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 369–76. New York: IEEE
30. DeCastro JA, Kress-Gazit H. 2016. Nonlinear controller synthesis and automatic workspace partitioning for reactive high-level behaviors. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pp. 225–34. New York: ACM
31. Johnson B, Kress-Gazit H. 2015. Analyzing and revising synthesized controllers for robots with sensing and actuation errors. *Int. J. Robot. Res.* 34:816–32
32. Ghosh S, Sadigh D, Nuzzo P, Raman V, Donzé A, et al. 2016. Diagnosis and repair for synthesis from signal temporal logic specifications. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pp. 31–40. New York: ACM
33. McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, et al. 1998. *PDDL—the Planning Domain Definition Language*. Tech. Rep. CVC TR-98-003/DCS TR-1165, Cent. Comput. Vision Control, Yale Univ., New Haven, CT
34. Bacchus F, Kabanza F. 1996. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 1215–22. Menlo Park, CA: AAAI Press
35. Giacomo GD, Vardi MY. 2000. Automata-theoretic approach to planning for temporally extended goals. In *ECP '99: Proceedings of the 5th European Conference on Planning*, pp. 226–38. Berlin: Springer
36. Schoppers M. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, ed. J McDermott, pp. 1039–46. Burlington, MA: Morgan Kaufmann

37. Pryor L, Collins G. 1996. Planning for contingencies: a decision-based approach. *J. Artif. Intell. Res.* 4:287–339
38. Heger FW, Singh S. 2010. Robust robotic assembly through contingencies, plan repair and re-planning. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3825–30. New York: IEEE
39. Cimatti A, Pistore M, Roveri M, Traverso P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.* 147:35–84
40. Cimatti A, Roveri M, Bertoli P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artif. Intell.* 159:127–206
41. Ehlers R, Lafortune S, Tripakis S, Vardi MY. 2017. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dyn. Syst. Theory Appl.* 27:209–60
42. Clarke EM, Grumberg O, Peled DA. 1999. *Model Checking*. Cambridge, MA: MIT Press
43. Rutten JJMM, Kwiatkowska M, Gethin N, Parker D. 2004. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*. Providence, RI: Am. Math. Soc.
44. Milner R. 1999. *Communicating and Mobile Systems: The π Calculus*. Cambridge, UK: Cambridge Univ. Press
45. Habets L, van Schuppen JH. 2004. A control problem for affine dynamical systems on a full-dimensional polytope. *Automatica* 40:21–35
46. Conner DC, Rizzi AA, Choset H. 2003. Composition of local potential functions for global robot control and navigation. In *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 4, pp. 3546–51. New York: IEEE
47. Lindemann SR, LaValle SM. 2005. Smoothly blending vector fields for global robot navigation. In *44th IEEE Conference on Decision and Control*, pp. 207–14. New York: IEEE
48. Rimon E, Koditschek DE. 1992. Exact robot navigation using artificial potential functions. *IEEE Trans. Robot. Autom.* 8:501–18
49. Ayanian N, Kumar V. 2010. Decentralized feedback controllers for multiagent teams in environments with obstacles. *IEEE Trans. Robot.* 26:878–87
50. Pola G, Girard A, Tabuada P. 2008. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica* 44:2508–16
51. Zamani M, Pola G, Mazo M, Tabuada P. 2012. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Trans. Autom. Control* 57:1804–9
52. Liu J, Ozay N. 2016. Finite abstractions with robustness margins for temporal logic-based control synthesis. *Nonlinear Anal. Hybrid Syst.* 22:1–15
53. Mazo M, Davitian A, Tabuada P. 2010. PESSOA: a tool for embedded controller synthesis. In *CAV 2010: Computer Aided Verification*, ed. T Touli, B Cook, P Jackson, pp. 566–69. Berlin: Springer
54. Mouelhi S, Girard A, Gössler G. 2013. CoSyMA: a tool for controller synthesis using multi-scale abstractions. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, pp. 83–88. New York: ACM
55. Rungger M, Zamani M. 2016. SCOTS: a tool for the synthesis of symbolic controllers. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pp. 99–104. New York: ACM
56. Mitchell IM, Bayen AM, Tomlin CJ. 2005. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Trans. Autom. Control* 50:947–57
57. Ding J, Gillula J, Huang H, Vitus MP, Zhang W, Tomlin CJ. 2011. Hybrid systems in robotics: toward reachability-based controller design. *IEEE Robot. Autom. Mag.* 18:33–43
58. Chen M, Tomlin CJ. 2015. Exact and efficient Hamilton-Jacobi reachability for decoupled systems. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 1297–303. New York: IEEE
59. Conner DC, Choset H, Rizzi AA. 2006. Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies. In *Robotics: Science and Systems II*, ed. GS Sukhatme, S Schaal, W Burgard, D Fox, chap. 8. Cambridge, MA: MIT Press
60. Tedrake R, Manchester IR, Tobenkin M, Roberts JW. 2010. LQR-trees: feedback motion planning via sums-of-squares verification. *Int. J. Robot. Res.* 29:1038–52
61. Tobenkin MM, Manchester IR, Tedrake R. 2011. Invariant funnels around trajectories using sum-of-squares programming. *IFAC Proc. Vol.* 44:9218–23

62. Majumdar A, Tobenkin M, Tedrake R. 2012. Algebraic verification for parameterized motion planning libraries. In *2012 American Control Conference (ACC)*, pp. 250–57. New York: IEEE
63. Majumdar A, Tedrake R. 2017. Funnel libraries for real-time robust feedback motion planning. *Int. J. Robot. Res.* 36:947–82
64. Tedrake R, Drake Dev. Team. 2014. *Drake: a planning, control, and analysis toolbox for nonlinear dynamical systems*. **http://drake.mit.edu**
65. DeCastro JA, Kress-Gazit H. 2015. Synthesis of nonlinear continuous controllers for verifiably correct high-level, reactive behaviors. *Int. J. Robot. Res.* 34:378–94
66. Raman V, Piterman N, Finucane C, Kress-Gazit H. 2015. Timing semantics for abstraction and execution of synthesized high-level robot control. *IEEE Trans. Robot.* 31:591–604
67. Sucan IA, Moll M, Kavraki LE. 2012. The open motion planning library. *IEEE Robot. Autom. Mag.* 19:72–82
68. Pnueli A, Rosner R. 1989. On the synthesis of a reactive module. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 179–90. New York: ACM
69. Bloem R, Jobstmann B, Piterman N, Pnueli A, Sa'ar Y. 2012. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.* 78:911–38
70. Kupferman O, Vardi MY. 2001. Model checking of safety properties. *Formal Methods Syst. Des.* 19:291–314
71. Baier C, Katoen JP. 2008. *Principles of Model Checking*. Cambridge, MA: MIT Press
72. Maler O, Nickovic D. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, ed. Y Lakhnech, S Yovine, pp. 152–66. Berlin: Springer
73. Raman V, Donzé A, Maasoumy M, Murray RM, Sangiovanni-Vincentelli AL, Seshia SA. 2014. Model predictive control with signal temporal logic specifications. In *2014 53rd IEEE Conference on Decision and Control (CDC)*, pp. 81–87. New York: IEEE
74. Fainekos GE, Pappas GJ. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.* 410:4262–91
75. Fainekos G, Kress-Gazit H, Pappas G. 2005. Temporal logic motion planning for mobile robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2020–25. New York: IEEE
76. Kloetzer M, Belta C. 2008. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. Autom. Control* 53:287–97
77. Lahijanian M, Kloetzer M, Itani S, Belta C, Andersson SB. 2009. Automatic deployment of autonomous cars in a robotic urban-like environment (RULE). In *2009 IEEE International Conference on Robotics and Automation*, pp. 2055–60. New York: IEEE
78. Fainekos GE, Girard A, Kress-Gazit H, Pappas GJ. 2009. Temporal logic motion planning for dynamic mobile robots. *Automatica* 45:343–52
79. Smith SL, Tumova J, Belta C, Rus D. 2011. Optimal path planning for surveillance with temporal-logic constraints. *Int. J. Robot. Res.* 30:1695–708
80. Wolff EM, Topcu U, Murray RM. 2013. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *2013 IEEE International Conference on Robotics and Automation*, pp. 5033–40. New York: IEEE
81. Maly MR, Lahijanian M, Kavraki LE, Kress-Gazit H, Vardi MY. 2013. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, pp. 353–62. New York: ACM
82. Lahijanian M, Maly MR, Fried D, Kavraki LE, Kress-Gazit H, Vardi MY. 2016. Iterative temporal planning in uncertain environments with partial satisfaction guarantees. *IEEE Trans. Robot.* 32:583–99
83. Bhatia A, Kavraki LE, Vardi MY. 2010. Sampling-based motion planning with temporal goals. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2689–96. New York: IEEE
84. Bhatia A, Maly MR, Kavraki LE, Vardi MY. 2011. Motion planning with complex goals. *IEEE Robot. Autom. Mag.* 18:55–64

85. Vasile CI, Belta C. 2013. Sampling-based temporal logic path planning. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4817–22. New York: IEEE
86. McMahon J, Plaku E. 2014. Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3726–33. New York: IEEE
87. He K, Lahijanian M, Kavraki LE, Vardi MY. 2015. Towards manipulation planning with temporal logic specifications. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 346–52. New York: IEEE
88. Lahijanian M, Wasniewski J, Andersson SB, Belta C. 2010. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3227–32. New York: IEEE
89. Lahijanian M, Andersson S, Belta C. 2011. Control of Markov decision processes from PCTL specifications. In *2011 American Control Conference*, pp. 311–16. New York: IEEE
90. Cizelj I, Ding XC, Lahijanian M, Pinto A, Belta C. 2011. Probabilistically safe vehicle control in a hostile environment. *IFAC-PapersOnline* 18:11803–8
91. Lahijanian M, Andersson SB, Belta C. 2012. Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Trans. Robot.* 28:396–409
92. Lahijanian M, Andersson S, Belta C. 2009. A probabilistic approach for control of a stochastic system from LTL specifications. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) Held Jointly with the 2009 28th Chinese Control Conference*, pp. 2236–41. New York: IEEE
93. Ding XC, Smith SL, Belta C, Rus D. 2011. LTL control in uncertain environments with probabilistic satisfaction guarantees. *IFAC-PapersOnline* 18:3515–20
94. Svorenova M, Cerna I, Belta C. 2013. Optimal control of MDPs with temporal logic constraints. In *2013 IEEE 52nd Annual Conference on Decision and Control (CDC)*, pp. 3938–43. New York: IEEE
95. Ding X, Smith SL, Belta C, Rus D. 2014. Optimal control of Markov decision processes with linear temporal logic constraints. *IEEE Trans. Autom. Control* 59:1244–57
96. Fu J, Topcu U. 2016. Synthesis of shared autonomy policies with temporal logic specifications. *IEEE Trans. Autom. Sci. Eng.* 13:7–17
97. Wang J, Ding XXC, Lahijanian M, Paschalidis IIC, Belta CCA, et al. 2012. Temporal logic motion control using actor-critic methods. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4687–92. New York: IEEE
98. Fu J, Topcu U. 2014. Probably approximately correct MDP learning and control with temporal logic constraints. In *Robotics: Science and Systems X*, ed. D Fox, LE Kavraki, H Kurniawati, chap. 39. N.p.: Robot. Sci. Syst. Found.
99. Brázdil T, Chatterjee K, Chmelík M, Forejt V, Křetínský J, et al. 2014. Verification of Markov decision processes using learning algorithms. In *Automated Technology for Verification and Analysis*, ed. F Cassez, JF Raskin, pp. 98–114. Berlin: Springer
100. Sadigh D, Kim ES, Coogan S, Sastry S, Seshia SA. 2014. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *IEEE 53rd Annual Conference on Decision and Control (CDC)*, pp. 1091–96. New York: IEEE
101. Wang J, Ding XXC, Lahijanian M, Paschalidis IIC, Belta CC. 2015. Temporal logic motion control using actor-critic methods. *Int. J. Robot. Res.* 34:1329–44
102. Wolff EM, Topcu U, Murray RM. 2012. Robust control of uncertain Markov decision processes with temporal logic specifications. In *2012 IEEE 51st Annual Conference on Decision and Control (CDC)*, pp. 3372–79. New York: IEEE
103. Lahijanian M, Andersson SB, Belta C. 2015. Formal verification and synthesis for discrete-time stochastic systems. *IEEE Trans. Autom. Control* 60:2031–45
104. Luna R, Lahijanian M, Moll M, Kavraki L. 2015. Asymptotically optimal stochastic motion planning with temporal goals. In *Algorithmic Foundations of Robotics XI*, pp. 335–52. Berlin: Springer
105. de Alfaro L. 1997. *Formal verification of probabilistic systems*. PhD Thesis, Dep. Comput. Sci., Stanford Univ., Stanford, CA
106. Kress-Gazit H, Fainekos GE, Pappas GJ. 2009. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Robot.* 25:1370–81

107. Kress-Gazit H, Wongpiromsarn T, Topcu U. 2011. Correct, reactive, high-level robot control. *IEEE Robot. Autom. Mag.* 18:65–74
108. Wongpiromsarn T, Topcu U, Murray RM. 2012. Receding horizon temporal logic planning. *IEEE Trans. Autom. Control* 57:2817–30
109. Kozen D. 1983. Results on the propositional μ-calculus. *Theor. Comput. Sci.* 27:333–54
110. Ehlers R. 2011. Generalized Rabin(1) synthesis with applications to robust system synthesis. In *NFM 2011: NASA Formal Methods*, ed. M Bobaru, K Havelund, GJ Holzmann, R Joshi, pp. 101–15. Berlin: Springer
111. Ehlers R, Raman V. 2016. Slugs: extensible GR(1) synthesis. In *CAV 2016: Computer Aided Verification*, ed. S Chaudhuri, A Farzan, pp. 333–39. Berlin: Springer
112. Pnueli A, Sa'ar Y, Zuck LD. 2010. JTLV: a framework for developing verification algorithms. In *CAV 2010: Computer Aided Verification*, ed. T Touili, B Cook, P Jackson, pp. 171–74. Berlin: Springer
113. Finucane C, Jing G, Kress-Gazit H. 2010. LTLMoP: experimenting with language, temporal logic and robot control. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1988–93. New York: IEEE
114. Filippidis I, Dathathri S, Livingston SC, Ozay N, Murray RM. 2016. Control design for hybrid systems with TuLiP: the Temporal Logic Planning toolbox. In *2016 IEEE Conference on Control Applications (CCA)*, pp. 1030–41. New York: IEEE
115. Karaman S, Sanfelice RG, Frazzoli E. 2008. Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *47th IEEE Conference on Decision and Control (CDC)*, pp. 2117–22. New York: IEEE
116. Kwon Y, Agha G. 2008. LTLC: linear temporal logic for control. In *HSCC '08: Proceedings of the 11th International Workshop on Hybrid Systems: Computation and Control*, pp. 316–29. Berlin: Springer
117. Wolff EM, Topcu U, Murray RM. 2014. Optimization-based trajectory generation with linear temporal logic specifications. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5319–25. New York: IEEE
118. Lindemann L, Dimarogonas DV. 2017. Robust motion planning employing signal temporal logic. In *2017 American Control Conference (ACC)*, pp. 2950–55. New York: IEEE
119. Gol EA, Lazar M. 2013. Temporal logic model predictive control for discrete-time systems. In *HSCC'13: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, pp. 343–52. New York: ACM
120. Raman V, Donzé A, Sadigh D, Murray RM, Seshia SA. 2015. Reactive synthesis from signal temporal logic specifications. In *HSCC '15: Proceedings of the 18th International Conference on Hybrid Systems Computation and Control*, pp. 239–48. New York: ACM
121. Sadigh D, Kapoor A. 2016. Safe control under uncertainty with probabilistic signal temporal logic. In *Robotics: Science and Systems XII*, ed. D Hu, N Amato, S Berman, S Jacobs, chap. 17. N.p.: Robot. Sci. Syst. Found.
122. Jha S, Raman V. 2016. Automated synthesis of safe autonomous vehicle control under perception uncertainty. In *NFM 2016: NASA Formal Methods*, ed. S Rayadurgam, O Tkachuk, pp. 117–32. Cham, Switz.: Springer