

Synthesis for Robots: Guarantees and Feedback for Robot Behavior

Hadas Kress-Gazit,¹ Morteza Lahijanian², and
Vasumathi Raman³

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

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

³San Francisco, CA, USA, 94103; email: vasu@cds.caltech.edu

Xxxx. Xxx. Xxx. Xxx. YYYY. AA:1–27

[https://doi.org/10.1146/\(\(please add article doi\)\)](https://doi.org/10.1146/((please add article doi)))

Copyright © YYYY by Annual Reviews.
All rights reserved

Keywords

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

Abstract

Robot control for tasks such as motion around obstacles or grasping objects has advanced significantly in the last few decades. However, controlling robots to perform complex tasks is largely still accomplished 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 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 paper 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 dual 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, that is, built in a way that is guaranteed to satisfy the requirements?”

The state of the art in robot control for tasks such as motion 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 largely still accomplished 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 providing feedback regarding the specifications themselves, such as whether they can or cannot 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 with a model of the environment, initial state and/or a cost function, and outputs either a controller or a proof that the specification is not feasible (is unrealizable). Roughly speaking, the synthesis techniques can be grouped into three types:

1. Open-loop (non-reactive): Given a robot model and a specification, find a sequence of states or actions that will guarantee the robot satisfies the specification. In this approach the environment is static and is typically not modeled.
2. 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.
3. 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 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 logic that 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 on robot behavior and assumption on environments (Section 3), and algorithmic approaches to synthesis (Section 4). The article focuses on single robot systems and temporal logic specifications; multi-robot systems (e.g. (3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)) and recent approaches such as synthesis through Satisfiability Modulo Theories (SMT) (e.g. (14, 15, 16)) are out of scope.

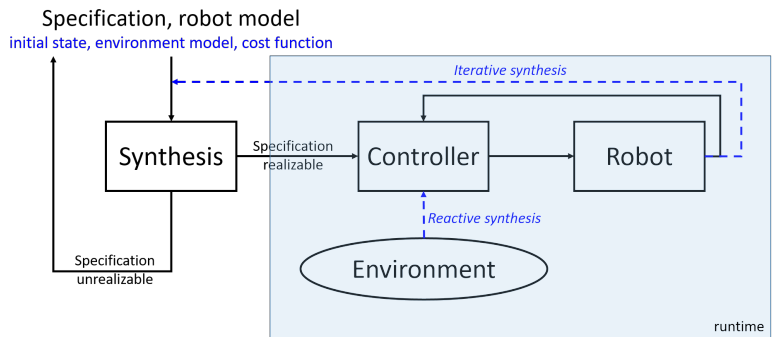


Figure 1: **Synthesis for robot control** : The input is a specification and a robot model. Some approaches also take as input the initial state of the robot, a model of the environment behavior and/or a cost function. The result of the synthesis algorithm is either a controller to be executed by the robot, or a proof that the specification is not feasible (is unrealizable). The black elements in the figure are common to all synthesis approaches while the blue elements are present in some of them.

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.

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 to manual composition of controllers.

Feedback and suggestions : Due 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 counter-example that shows under what conditions the robot will fail. Leveraging these counter-examples, 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 of the specifications themselves which are often written by humans, and hence, 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 a tautology (17, 20), i.e. where a controller can be created that may not do anything. In the context of suggesting changes, (21, 22, 23, 24, 25, 26) explore “minimum distance” revisions for LTL and automata-based specifications. Methods for generating additional symbolic environment assumptions that would make the specification realizable are explored in (27, 28). Automatic revisions to the abstractions based on the dynamics of the robot are presented in (29, 30), and suggested revisions due to probabilistic analysis of a synthesized controller are discussed in (31, 26). 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 paper, 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 (AI) planning community and the Discrete Event Systems (DES) community.

In AI, 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 to the goal state. Variants that are closer to the work described in this paper 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)) that synthesize reaction rules for possible environment behaviors, and contingency planners (e.g. (37, 38)) that 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/planning problem can and cannot be solved.

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

1.3. Example

The following example is used to illustrate the concepts discussed in this paper. Consider a mobile robot moving in the workspace depicted in Fig 2. The workspace contains areas

of interest, such as 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 paper is a mail-delivery task, where the robot is instructed to deliver letters and packages to rooms and/or people.

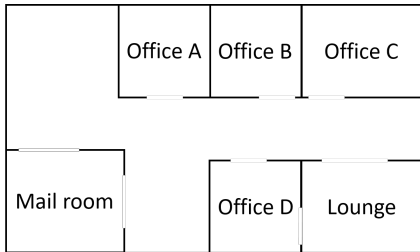


Figure 2: Workspace for mail delivery example

2. Abstractions: System representations

The synthesis algorithms described in this paper 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 that can be 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 $k > 0$, $t_{k+1} - t_k = \Delta t$.

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

Trajectory ξ :
Execution of the system dynamics model

2.2. Symbolic representations

Synthesis techniques that are covered in this paper are based on deterministic, non-deterministic 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 (K) 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 $(s, s') \in R$,
- $L : S \rightarrow 2^{AP}$ 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 $\omega = \omega_0 \rightarrow \omega_1 \rightarrow \omega_2 \cdots$ where $\omega_0 \in S_0$, $\omega_i \in S$, and $(\omega_i, \omega_{i+1}) \in R$ for all $i \geq 0$. 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 ,
- $Steps : S \times Act \rightarrow Dist(S)$ 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,
- $L : S \rightarrow 2^{AP}$ 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 $i \geq 0$, $\omega_i \in S$, $a_{i+1} \in A(\omega_i)$ 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 only depends on $last(\omega^{fin})$, 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 the ability of the robot to continuously implement all the symbolic transitions in the controller. This is formally defined as a simulation relation (44) where one system can mimic all the behaviors of the second system. For robots, ideally, the

Path ω : Path in symbolic structure

Trace/Word σ : Sequence of labels

$Path_s, Path_s^{fin}$: Set of infinite and finite paths starting in state s , respectively

Control policy μ : Function mapping finite paths to actions

continuous physical system simulates the symbolic one.

The following subsections 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 assigns physical meaning to them. This section describes the creation of abstractions relating to the robot motion and action in the workspace. For abstracting sensor information or more generally the environment state, one can use classifiers or other perception algorithms which are out of scope of this paper.

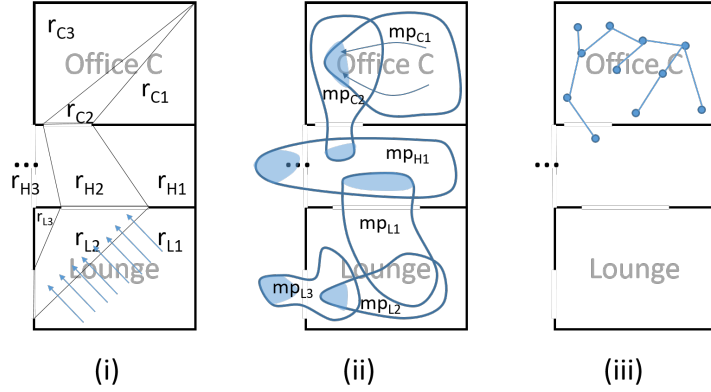
The abstraction algorithms are grouped into three categories: partitions (Section 2.3.1) where the labels are mutually exclusive and the continuous system simulates the symbolic, motion primitives (Section 2.3.2) where the labels may overlap and the continuous system simulates the symbolic, and motion planners (Section 2.3.3) where the labels are typically mutually exclusive but the transitions are probabilistically complete, meaning 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 non-overlapping 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.

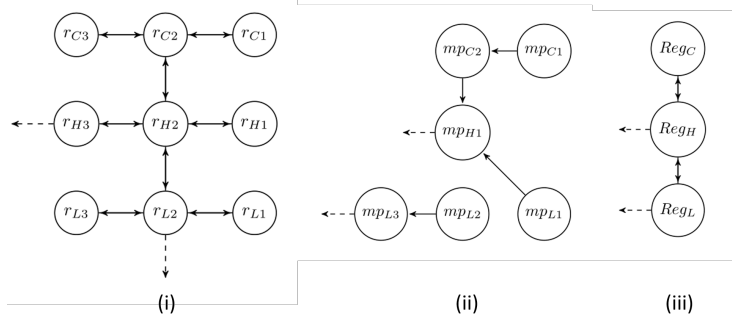
2.3.1. Partitions. In this approach, the continuous state space of the robot $X \subseteq \mathbb{R}^{n_c}$ is partitioned into a set $\{r_i\}$ such that $r_i \subseteq X$, $\cup_i r_i = X$ and $r_i \cap r_j = \emptyset$ for all $i \neq j$. Overloading notation, the symbol r_i is used as a label in the symbolic structure to mean that when a node is labeled with r_i , the physical system's state $x \in r_i \subseteq X$. Due to the partition, the symbols r_i 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, i.e. if cells are adjacent, there may not be a transition in the symbolic structure. Figure 3, left column depicts a possible partition for a subset of the workspace of the mail-delivery example and the corresponding symbolic structure.

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 (45, 46, 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, other potential field based controllers such as navigation functions (48) can be used to create the symbolic structure. For multi robot 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 non-deterministic structure that takes into account the effects of discretization. These approaches typically compute, for each cell in the grid, an over approximation 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)). Robustness of such abstractions to phenomena such as delays, measurement errors and model uncertainties is discussed in (52). 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, 54, 55).



(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 outline of motion primitive mp_i represent its invariant set r_i and the blue region represent its goals set g_i , and (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, 46, 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 $g_i \subseteq r_j$, and (iii) motion planner - the symbolic structure is initially fully connected (includes bidirectional arrows between adjacent regions). If the motion planner fails to find a path between regions i and j , the arrow connecting Reg_i and Reg_j is removed.

Figure 3: Abstractions and symbolic structures for the mail delivery example.

2.3.2. Motion primitives. Similar to the partition approach, the physical meaning of the symbols in the abstraction using motion primitives are related to regions of the state space. The main differences are that (i) the state space is no longer divided into a grid, (ii) the set of regions representing motion primitives are usually not disjoint, and (iii) 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 $r_i \subseteq X$. 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 notation, the symbol r_i is used 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 exist, then motion primitive j can be activated after the completion of motion primitive i , but the reverse does not hold. Figure 3, middle column depicts a possible set of motion primitives for part of the workspace for the example and the corresponding symbolic structure.

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 primitive is pervasive in robotics, with many different approaches to generating them such as reinforcement learning, learning by demonstration, control theory, and more. In order to enable guarantees for the robot’s physical behavior, a simulation relation is required 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** (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, 57, 58), creating vector fields over regions in the environment (46, 59), and Sums-of-Squares optimization for generating funnels around trajectories (60, 61, 62, 63, 64).

In the case of reactive synthesis, where the behavior of the robot might change due to changes in the environment, to ensure correct execution with motion primitives the abstraction must be reactively composable (65, 30). 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, special care with respect to timing semantics is needed when designing abstractions for robots with multiple actuation capabilities (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, together with 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, reachability computations similar to the ones discussed in Section 2.3.1, etc.

Similar to the partition approach, the workspace is discretized into cells; however, as opposed 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 $\cup_i Reg_i = \mathcal{W}$. Overloading notation, Reg_i is the symbol that is true when the robot is in region i of the partition. The transitions correspond to adjacency of the regions, i.e., if regions share a boundary then there exists a bi-directional transition between these regions, unless such a transition was removed during the iterative synthesis procedure. Figure 3, right column depicts the symbolic structure and a motion planner searching for a transition.

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 due to 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 employed 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 – the “what”, to an implementation – the “how”. Specifications can be roughly grouped into two types: **safety** specifications, which describe how the robot should always behave, e.g. it should “never collide with an obstacle” or “always maintain line-of-sight from a base station”, and **liveness** specifications, which describe goals or tasks/states the robot must eventually achieve, e.g. it should “eventually go back to the recharge station” or “once in a while send its location”. Safety and liveness specifications can be bounded over finite or infinite horizons, be deterministic or probabilistic, and 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 on 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 due 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 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 Linear Temporal Logic (LTL), described below. It is worth noting that there are two notation conventions in the literature for the temporal operators; either \bigcirc , \square , \diamond , \mathcal{U} , or X , G , F , U . In this paper, the former convention is followed.

3.1.1. LTL 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”: $\diamond\varphi = \text{True} \mathcal{U} \varphi$ and “always”: $\square\varphi = \neg \diamond \neg \varphi$.

3.1.2. LTL Semantics. The semantics of an LTL formula φ are defined on an infinite sequence $\sigma = \sigma_1\sigma_2\dots$ 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:

- $\sigma, i \models \pi$ 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 $\sigma, i+1 \models \varphi$,
- $\sigma, i \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists $k \geq i$ such that $\sigma, k \models \varphi_2$, and for all $i \leq j < k$, $\sigma, j \models \varphi_1$.

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 formula $\square\varphi$ if φ is True in every position of the sequence, and satisfies the formula $\diamond\varphi$ if φ is True at some position of the sequence.

3.1.3. Fragments of LTL. As will be 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) fragment (69) and co-safe LTL (70).

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 . It is important to note that one does not have to make any assumptions about the environment; specifying $\varphi_e = \text{True}$ 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 following structure

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

where

- φ_i^e and φ_i^s are non-temporal Boolean formulas constraining (if at all) the initial value(s) of the sensor propositions \mathcal{X} and robot propositions \mathcal{Y} respectively.

- φ_i^e and φ_i^s represents **safety** assumptions and requirements, i.e. constraints that must always hold, on 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_i^e and φ_i^s includes motion constraints, for example, if the robot is in Office A, in the next state it can only be in either Office A or the Hallway, and behavior requirements, for example, if the robot is carrying a package it may not go into the Lounge.
- 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 .

Co-safe LTL: 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 the operator “always” (\square) is not part of the fragment (the truth value of a formula with \square can only be evaluated over infinite traces of a system), while “eventually” (\diamond) is.

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. PCTL Syntax. As with LTL, formulas are defined over a set of atomic propositions AP . PCTL formulas are state formulas defined recursively as:

$$\begin{aligned} \text{State formulas:} \quad \varphi &::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathcal{P}_{\bowtie p}[\psi] \\ \text{Path formulas:} \quad \psi &::= \bigcirc\varphi \mid \varphi \mathcal{U}^{\leq k} \varphi \mid \varphi \mathcal{U} \varphi \end{aligned}$$

where \mathcal{P} is the probabilistic operator, $\bowtie \in \{<, \leq, \geq, >\}$, $p \in [0, 1]$, 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 only allowed 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 (\neg) and disjunction (\vee), and “eventually” (\diamond) can be derived from “until” (\mathcal{U}). By using \diamond , the “always” (\square) operator can be defined as $\mathcal{P}_{\bowtie p}[\square\varphi] \equiv \mathcal{P}_{\bowtie p}[\diamond\neg\varphi]$, where $\lesseqgtr \equiv \geq$, $\lessseqgtr \equiv >$, $\gtrless \equiv <$, and $\gtrless \equiv \leq$ (71). Similarly, the bounded operators $\diamond^{\leq k}$ and $\square^{\leq k}$ can be defined using $\mathcal{U}^{\leq k}$.

3.2.2. PCTL Semantics. PCTL formulas can be evaluated either over discrete-time Markov chains or Markov Decision Processes (MDP). In this paper, 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 $s \models \varphi_2$,
- $s \models \mathcal{P}_{\bowtie p}[\psi]$ 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 policy μ .

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

- $\omega \models \bigcirc\varphi$ iff $\omega_1 \models \varphi$,
- $\omega \models \varphi_1 \mathcal{U}^{\leq k} \varphi_2$ iff there exists $i \leq k$ such that $\omega_i \models \varphi_2$ and for all $j < i$, $\omega_j \models \varphi_1$,
- $\omega \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists $k \geq 0$ such that $\omega \models \varphi_1 \mathcal{U}^{\leq k} \varphi_2$.

3.3. Metric logics

The logics described above are defined over propositions that are Boolean variables that can either be 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. STL Syntax. Let AP^γ be a set of atomic predicates where $\pi^\gamma \in AP^\gamma$ is a predicate $X \rightarrow \{0, 1\}$ whose truth value corresponds to the sign of the function $\gamma : \mathbb{R}^{n_c} \rightarrow \mathbb{R}$. Signal Temporal Logic (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 $\langle \in \{[, (, \rangle, \} \}$ and $a, b \in \mathbb{R}_{\geq 0}$. 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 predicate that replaces propositions, that is, 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. STL Semantics. The satisfaction of an STL formula φ at time t is defined as:

- $(x, t_k) \models \pi^\gamma$ 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 $(x, t_k) \models \varphi_2$,
- $(x, t_k) \models \varphi_1 \mathcal{U}_{\langle a, b \rangle} \varphi_2$ 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, $\mathbf{x} = x_0 x_1 x_2 \dots$, satisfies φ , denoted by $\mathbf{x} \models \varphi$, if $(\mathbf{x}, t_0) \models \varphi$. Informally, $\mathbf{x} \models \Box_{[a, b]} \varphi$ if φ holds for all time between a and b , and $\mathbf{x} \models \varphi \mathcal{U}_{[a, b]} \psi$ if φ holds at every time step before ψ holds, and ψ 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 which, in addition to the yes/no answer to the satisfaction question, provides a real number $\rho^\varphi(x, t)$, called the robustness of satisfaction, grading 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 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 (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 a package is available. The set of functions γ_{Region} 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 (non-reactive): $(\neg Deliver)U(Deliver \wedge (Office_B \vee Office_C))$ expresses “deliver a package to one of offices B or C but not anywhere else” .

GR(1) fragment (reactive): $\Box \Diamond (SensePackage \rightarrow (Deliver \wedge (Office_B \vee Office_C))) \wedge \Box \Diamond (\neg SensePackage \rightarrow Mailroom)$ 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 C with 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, this section gives an overview of the synthesis algorithms used to synthesize controllers. For each algorithm, the margin notes describe the abstraction, specification formalisms and structure of the resulting controller relevant to that algorithm.

4.1. Automata based

Automata-based synthesis algorithms are defined for systems that are abstracted as a Kripke structure \mathcal{K} . Here the specifications are given as LTL formulas and the abstraction \mathcal{K} is deterministic, i.e., 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 non-deterministic systems are discussed in Sections 4.2 and 4.3.

Automata-based synthesis methods generally include three main steps:

1. translation of the specification into an automaton,
2. composition of the system abstraction with the automaton, and
3. 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,

Abstraction:
Deterministic Kripke structure

Specification: LTL

Synthesis product:
Sequence of states

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 $\delta : Z \times \Sigma \rightarrow 2^Z$, and a set of accepting states $F^{\text{NBA}} \subseteq Z$. 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 $F^{\mathcal{P}} = S \times F^{\text{NBA}}$ 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: (i) identifying the cycles in $\mathcal{K}^{\mathcal{P}}$ that contain at least one accepting state in $F^{\mathcal{P}}$, and (ii) finding a path from an initial state to one of the cycles. This results in a path $\omega^{\mathcal{P}} = \underline{\omega}^{\mathcal{P}} \bar{\omega}^{\mathcal{P}}$, where $\underline{\omega}^{\mathcal{P}}$ is a finite path, known as the prefix, and $\bar{\omega}^{\mathcal{P}}$ 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 $F^{\mathcal{P}} = S \times F^{\text{DFA}}$ 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 follow the general steps of the algorithm described above, e.g., (75, 76, 77, 78, 79, 80, 81, 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, (76) introduces 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 are 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 (75), instead of explicitly constructing the specification automaton, model checkers are 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., (83, 84, 85, 81, 86, 87, 82). With these techniques, though, 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. MDP based

In MDP-based synthesis, the focus is on generating a policy that maximizes (or in some cases minimizes) the probability of satisfying the specification. The first work (88, 89, 90, 91) in

MDP-based synthesis focused on specifications given as probabilistic logic formulas, namely PCTL (43). These 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, though, 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, 94, 95, 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, 98, 99, 100, 101). In recent years, synthesis methods for uncertain MDPs have been studied to relax the single-valued transition probability of MDPs (102, 103, 104). By allowing uncertainty over the transition probabilities, these models arguably provide a better modeling framework for physical systems with noise than classical MDPs since it is typically difficult to compute an exact transition probability for such systems.

PCTL Synthesis for MDPs: 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.

Table 1: Probability optimization formulations for MDP-based synthesis.

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

Abstraction: MDP
Specification: PCTL
Synthesis product:
 Policy mapping
 finite paths to an
 action

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 1, where \mathbf{p}_s^* denotes the optimal probability of satisfying φ at the state $s \in S$, and $\text{Sat}(\varphi_1) \subseteq S$ 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 1, which can be computed by k matrix-vector multiplications. This results in a time dependent policy, i.e., 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 1. 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 $s \in S$.

LTL Synthesis for MDPs: 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 (DRA) (71) is generated from the LTL formula φ . The product $\mathcal{M}^{\mathcal{P}} = \mathcal{M} \times \mathcal{A}$ is then generated similar 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: $Steps^{\mathcal{P}}((s, z), a^{\mathcal{P}})((s', z')) = Steps(s, a)(s')$ 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 DRA 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 is shown in Table 1. 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 .

Abstraction: MDP
Specification: LTL
Synthesis product: Policy mapping finite paths to an action

4.3. Game based

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 been proven to have 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 (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_s^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, 107, 108).

Synthesis from GR(1) specifications reduces to solving a μ -calculus fixpoint equation with three nested fixpoints 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 (109, 69); an informal description of the relevant portions are summarized as follows

Abstraction: Robot and environment transitions are part of the specification

Specification: LTL, GR(1) fragment of LTL

Synthesis product: Finite state controller

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

In (69), the set of winning states for the robot is characterized by the μ -calculus formula $\varphi_{win} =$

$$\nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_n \end{bmatrix} \cdot \begin{bmatrix} \mu Y. (\bigvee_{i=1}^m \nu X. (J_s^1 \wedge \otimes Z_2 \vee \otimes Y \vee \neg J_e^i \wedge \otimes X)) \\ \mu Y. (\bigvee_{i=1}^m \nu X. (J_s^2 \wedge \otimes Z_3 \vee \otimes Y \vee \neg J_e^i \wedge \otimes X)) \\ \vdots \\ \mu Y. (\bigvee_{i=1}^m \nu X. (J_s^n \wedge \otimes Z_1 \vee \otimes Y \vee \neg J_e^i \wedge \otimes X)) \end{bmatrix} \quad (1)$$

where J_e^i is the i^{th} environment liveness from φ_e^g ($i \in \{1, \dots, m\}$), and J_s^j is the j^{th} robot liveness from φ_s^g ($j \in \{1, \dots, n\}$). Let \oplus denote summation modulo n . For $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$, the greatest fixpoint $\nu X. (J_s^j \wedge \otimes Z_{j \oplus 1} \vee \otimes Y \vee \neg J_e^i \wedge \otimes X)$ 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 $Q_{win} = \llbracket J_s^j \wedge \otimes Z_{j \oplus 1} \vee \otimes Y \rrbracket$. The two outer fixpoints ensure that the robot wins from the set Q_{win} : μY ensures that the play reaches a $J_s^j \wedge \otimes Z_{j \oplus 1}$ 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 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 (110, 111, 66). Moreover, open source tools such as JTLV (112) and `slugs` (111) do not build the game structure explicitly but use binary decision diagrams (BDDs) as a symbolic data structure to efficiently solve (1). Robotics and control-specific tools such as LTLMoP (113) and TuLIP (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

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 $x_0 \in X$, 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 (formal summary in Table 2):

Open-loop (non-reactive): Find a control signal \mathbf{u} of length N given a nominal environment \mathbf{w} (115, 116, 73, 117, 118). The state of the robot is assumed fully observable, and the environment inputs are known in advance. To allow 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 (MPC) 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 (119, 73, 120). In (73, 120), a control input is 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 (120), counterexamples are used to inductively refine the synthesized controller until convergence or a maximum number of iterations.

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

Table 2: Optimization-based synthesis formulations

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

All 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, (73) show how the robustness of a STL specification φ can be recursively encoded using Mixed Integer-Linear Program (MILP) constraints, and enforcing $\rho^\varphi(x, t) > 0$ ensures satisfaction of the formula. In (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-

Abstraction:

Difference equation, polytopic regions of interest

Specification: LTL, MTL, or STL, and a cost function J

Synthesis product: Control sequence

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.

Mixed integer-linear programs 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 describes formal synthesis for robot control – the models used to define the problem, the specification formalisms used to capture high-level, complex 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 as 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. Synthesis’ 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 high potential impact but the techniques are not widely used yet. 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 the abstraction is too coarse) to enable a simulation relation with the continuous physical system. Existing software for control and actuation of robots are typically not 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 to see synthesis become an important tool in the robust and reliable deployment of robotic systems.

FUTURE DIRECTIONS

1. Abstractions: The choice of abstraction level impacts both the scalability of synthesis and the guarantees it can provide. Too fine grained an abstraction causes synthesis to become intractable, too coarse an abstraction causes the models to

lose fidelity with respect to the physical system and 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, for scaling up synthesis, for creating abstractions that can be used for synthesis, and for creating explainable AI. The challenge is maintaining the guarantees while leveraging data-driven approaches to control.
3. Synthesis and Human-Robot Interaction (HRI): Formalizing models and creating abstractions and specifications for HRI will enable synthesizing robot controllers and feedback that are task, environment and also interaction dependent thus creating robots that can explain and guarantee their behavior in an HRI 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 Computing Surveys* 28:626–643
2. Spenko M, Buerger S, Iagnemma K. 2017. Special Issue: The DARPA Robotics Challenge Finals. *Journal of Field Robotics* 34:225–426
3. Loizou SSG, Kyriakopoulos KJ. 2004. Automatic synthesis of multiagent motion tasks based on LTL specifications. In *IEEE Conf. on Decision and Control*. Bahamas
4. Kloetzer M, Belta C. 2007. Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions. *{IEEE} Transactions on Robotics* 23:320–330
5. Karaman S, Frazzoli E. 2008. Complex Mission Optimization for Multiple-UAVs using Linear Temporal Logic. In *American Control Conference*. Seattle, Washington
6. Kress-Gazit H, Ayanian N, Pappas GJ, Kumar V. 2008. Recycling controllers. In *IEEE Conference on Automation Science and Engineering*. Washington D.C., USA
7. Filippidis I, Dimarogonas DV, Kyriakopoulos KJ. 2012. Decentralized multi-agent control from local LTL specifications. In *IEEE Conference on Decision and Control*
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 *Robotics and Automation (ICRA), 2013 IEEE International Conference on*
9. Raman V, Kress-Gazit H. 2014. Synthesis for multi-robot controllers with interleaved motion. In *Proceedings - IEEE International Conference on Robotics and Automation*
10. Raman V. 2014. Reactive switching protocols for multi-robot high-level tasks. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*
11. DeCastro JA, Alonso-mora J, Raman V, Rus D. 2015. Collision-Free Reactive Mission and Motion Planning for Multi-Robot Systems. In *International Symposium on Robotics Research (ISRR)*. Sestri Levante, Italy
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)*

13. Tumova J, Dimarogonas DV. 2016. Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica* 70:239–248
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 *Int. Conf. on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*
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)*. IEEE
16. Shoukry Y, Nuzzo P, Saha I, Sangiovanni-Vincentelli AL, Seshia SA, et al. 2016. Scalable lazy SMT-based motion planning. In *55th {IEEE} Conference on Decision and Control, {CDC} 2016, Las Vegas, NV, USA, December 12-14, 2016*
17. Raman V, Kress-Gazit H. 2013. Explaining impossible high-level robot behaviors. *IEEE Transactions on Robotics* 29:94–104
18. Lignos C, Raman V, Finucane C, Marcus M, Kress-Gazit H. 2014. Provably correct reactive control from natural language. *Autonomous 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 Dynamic Systems: Theory and Applications* 27:371–405
20. Dokhanchi A, Hoxha B, Faïnekos G. 2015. Metric interval temporal logic specification elicitation and debugging. In *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*. IEEE
21. Faïnekos GE. 2011. Revising Temporal Logic Specifications for Motion Planning. In *Proceedings of the IEEE Conference on Robotics and Automation*
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*
23. Guo M, Johansson KH, Dimarogonas DV. 2013. Revising motion planning under Linear Temporal Logic specifications in partially known workspaces. In *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE
24. Kim K, Faïnekos G, Sankaranarayanan S. 2015. On the minimal revision problem of specification automata. *The International Journal of Robotics Research* 34:1515–1535
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 *The Twenty-Ninth AAAI Conference (AAAI-15)*. Austin, TX: AAAI
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, CDC 2016*. IEEE
27. Li W, Dworkin L, Seshia SA. 2011. Mining assumptions for synthesis. In *9th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE 2011*
28. Alur R, Moarref S, Topcu U. 2013. Counter-strategy guided refinement of GR(1) temporal logic specifications. In *Formal Methods in Computer-Aided Design (FMCAD 2013)*
29. DeCastro JA, Raman V, Kress-Gazit H. 2015. Dynamics-driven adaptive abstraction for reactive high-level mission and motion planning. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*
30. DeCastro JA, Kress-Gazit H. 2016. Nonlinear Controller Synthesis and Automatic Workspace Partitioning for Reactive High-Level Behaviors. In *Hybrid Systems: Computation and Control, Vienna, Austria, April 12-14, 2016*
31. Johnson B, Kress-Gazit H. 2015. Analyzing and revising synthesized controllers for robots with sensing and actuation errors. *International Journal of Robotics Research* 34:816–832
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, HSCC 2016, Vienna, Austria, April 12-14,*

33. McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, et al. 1998. {PDDL} – {The Planning Domain Definition Language}. Tech. rep., Yale Center for Computational Vision and Control
34. Bacchus F, Kabanza F. 1996. Planning for Temporally Extended Goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence ({AAAI}-96)*. Portland, Oregon, USA: AAAI Press / The MIT 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*. London, UK: Springer-Verlag
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. Milan, Italy: Morgan Kaufmann publishers Inc.
37. Pryor L, Collins G. 1996. Planning for Contingencies: {A} Decision-based Approach. *J. Artif. Intell. Res. ({JAIR})* 4:287–339
38. Heger FW, Singh S. 2010. Robust robotic assembly through contingencies, plan repair and re-planning. In *{IEEE} International Conference on Robotics and Automation, {ICRA} 2010, Anchorage, Alaska, USA, 3-7 May 2010*
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 Dynamic Systems* 27:209–260
42. Clarke EM, Grumberg O, Peled DA. 1999. Model Checking. Cambridge, Massachusetts: MIT Press
43. Rutten JJMM, Kwiatkowska M, Gethin N, Parker D. 2004. Mathematical techniques for analyzing concurrent and probabilistic systems. American Mathematical Society
44. Milner R. 1999. Communicating and mobile systems : the pi-calculus. Cambridge University 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 *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, vol. 4. Carnegie Mellon University, Robotics Institute, Las Vegas, NV
47. Lindemann SR, LaValle SM. 2005. Smoothly blending vector fields for global robot navigation. In *IEEE Conference on Decision and Control*. Seville, Spain
48. Rimón E, Koditschek DE. 1992. Exact Robot Navigation Using Artificial Potential Functions. *IEEE Transactions on Robotics and Automation* 8:501–518
49. Ayanian N, Kumar V. 2010. Decentralized Feedback Controllers for Multiagent Teams in Environments With Obstacles. *{IEEE} Transactions on Robotics* 26:878–887
50. Pola G, Girard A, Tabuada P. 2008. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica* 44:2508–2516
51. Zamani M, Pola G, Mazo M, Tabuada P. 2012. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Transactions on Automatic Control* 57:1804–1809
52. Liu J, Ozay N. 2016. Finite abstractions with robustness margins for temporal logic-based control synthesis. *Nonlinear Analysis: Hybrid Systems* 22:1–15
53. Mazo M, Davitian A, Tabuada P. 2010. PESSOA: A Tool for Embedded Controller Synthesis. In *22nd International Conference on Computer Aided Verification (CAV 2010)*
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 - HSCC '13*. New York, New York, USA: ACM Press

55. Rungger M, Zamani M. 2016. SCOTS : A Tool for the Synthesis of Symbolic Controllers. In *Hybrid Systems : Computation and Control*
56. Mitchell IM, Bayen AM, Tomlin CJ. 2005. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Trans. Automat. Contr.* 50:947–957
57. Ding J, Gillula J, Huang H, Vitus MP, Zhang W, Tomlin CJ. 2011. Hybrid Systems in Robotics: Toward Reachability-Based Controller Design. *IEEE Robotics & Automation Magazine* 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)*. 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*. Philadelphia, PA
60. Tedrake R, Manchester IR, Tobenkin M, Roberts JW. 2010. LQR-trees: Feedback Motion Planning via Sums-of-Squares Verification. *Int. J. of Robotics Research* 29:1038–1052
61. Tobenkin MM, Manchester IR, Tedrake R. 2011. Invariant Funnels around Trajectories using Sum-of-Squares Programming. In *Proceedings of the 18th World Congress The International Federation of Automatic Control*. IFAC
62. Majumdar A, Tobenkin M, Tedrake R. 2012. Algebraic Verification for Parameterized Motion Planning Libraries. In *Proc. of the 2012 American Control Conference (ACC)*
63. Majumdar A, Tedrake R. 2017. Funnel Libraries for Real-Time Robust Feedback Motion Planning. *International Journal of Robotics Research*
64. Tedrake R, the Drake Development Team. 2014. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems
65. DeCastro JA, Kress-Gazit H. 2015. Synthesis of nonlinear continuous controllers for verifiably correct high-level, reactive behaviors. *International Journal of Robotics Research* 34:378–394
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. Robotics* 31:591–604
67. Sucan IA, Moll M, Kavraki LE. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 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*. ACM Press
69. Bloem R, Jobstmann B, Piterman N, Pnueli A, Sa'Ar Y. 2012. Synthesis of Reactive(1) designs. *Journal of Computer and System Sciences* 78:911–938
70. Kupferman O, Vardi MY. 2001. Model Checking of Safety Properties. *Formal Methods in System Design* 19:291–314
71. Baier C, Katoen JP. 2008. Principles Of Model Checking, vol. 950. 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: Joint International Conferences on Formal Modeling and Analysis of Timed Systems*
73. Raman V, Donzé A, Maasoumy M, Murray RM, Sangiovanni-Vincentelli AL, Seshia SA. 2014. Model predictive control with signal temporal logic specifications. In *53rd {IEEE} Conference on Decision and Control, {CDC} 2014, Los Angeles, CA, USA, December 15-17, 2014*
74. Fainekos GE, Pappas GJ. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.* 410:4262–4291
75. Fainekos G, Kress-Gazit H, Pappas G. 2005. Temporal Logic Motion Planning for Mobile Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*
76. Kloetzer M, Belta C. 2008. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control* 53:287–297
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 *Proceedings - IEEE Inter-*

- national Conference on Robotics and Automation*
78. Fainekos GE, Girard A, Kress-Gazit H, Pappas GJ. 2009. Temporal Logic Motion Planning for Dynamic Mobile Robots. *Automatica* 45:343–352
 79. Smith SL, Tumova J, Belta C, Rus D. 2011. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research* 30:1695–1708
 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, Karlsruhe, Germany, May 6-10, 2013*
 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 *Hybrid Systems: Computation and Control*
 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 Transactions on Robotics* 32:583–599
 83. Bhatia A, Kavraki LE, Vardi MY. 2010. Sampling-based motion planning with temporal goals. In *Proceedings - IEEE International Conference on Robotics and Automation*
 84. Bhatia A, Maly MR, Kavraki LE, Vardi MY. 2011. Motion Planning with Complex Goals. *IEEE Robotics & Automation Magazine* 18:55–64
 85. Vasile CI, Belta C. 2013. Sampling-based temporal logic path planning. In *IEEE International Conference on Intelligent Robots and Systems*
 86. McMahon J, Plaku E. 2014. Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic. In *Int. Conf. on Intelligent Robots and Systems*
 87. He K, Lahijanian M, Kavraki LE, Vardi MY. 2015. Towards manipulation planning with temporal logic specifications. In *Int. Conf. on Robotics and Automation*. IEEE
 88. Lahijanian M, Wasniewski J, Andersson SB, Belta C. 2010. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *Proceedings - IEEE International Conference on Robotics and Automation*
 89. Lahijanian M, Andersson S, Belta C. 2011. Control of Markov decision processes from PCTL specifications. In *Proceedings of the American Control Conference*
 90. Cizelj I, Ding XC, Lahijanian M, Pinto A, Belta C. 2011. Probabilistically safe vehicle control in a hostile environment. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 18
 91. Lahijanian M, Andersson SB, Belta C. 2012. Temporal Logic Motion Planning and Control With Probabilistic Satisfaction Guarantees. *{IEEE} Trans. Robotics* 28:396–409
 92. Lahijanian M, Andersson S, Belta C. 2009. A probabilistic approach for control of a stochastic system from LTL specifications. *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference :2236–2241*
 93. Ding XC, Smith SL, Belta C, Rus D. 2011. LTL control in uncertain environments with probabilistic satisfaction guarantees. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 18
 94. Svorenova M, Cerna I, Belta C. 2013. Optimal control of MDPs with temporal logic constraints. In *Proceedings of the IEEE Conference on Decision and Control*
 95. Ding X, Smith SL, Belta C, Rus D. 2014. Optimal control of Markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control* 59:1244–1257
 96. Fu J, Topcu U. 2016. Synthesis of shared autonomy policies with temporal logic specifications. *IEEE Transactions on Automation Science and Engineering* 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 *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 34
 98. Fu J, Topcu U. 2014. Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints. In *Robotics: Science and Systems X*. Robotics: Science and Systems Foundation
 99. Brázdil T, Chatterjee K, Chmelík M, Forejt V, Křetínský J, et al. 2014. Verification of Markov

- decision processes using learning algorithms. In *Proc. 12th International Symposium on Automated Technology for Verification and Analysis (ATVA'14)*, eds. F Cassez, JF Raskin, vol. 8837 of *LNCS*. 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)*. IEEE
 101. Wang J, Ding XXC, Lahijanian M, Paschalidis IIC, Belta CC. 2015. Temporal logic motion control using actor-critic methods. *International Journal of Robotics Research* 34:1329–1344
 102. Wolff EM, Topcu U, Murray RM. 2012. Robust control of uncertain markov decision processes with temporal logic specifications. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE
 103. Lahijanian M, Andersson SB, Belta C. 2015. Formal Verification and Synthesis for Discrete-Time Stochastic Systems. *IEEE Transactions on Automatic Control* 60:2031–2045
 104. Luna R, Lahijanian M, Moll M, Kavraki L. 2015. Asymptotically optimal stochastic motion planning with temporal goals, vol. 107. Springer
 105. De Alfaro L. 1997. Formal verification of probabilistic systems. Ph.D. thesis
 106. Kress-Gazit H, Fainekos GE, Pappas GJ. 2009. Temporal-Logic-Based Reactive Mission and Motion Planning. *{IEEE} Trans. Robotics* 25:1370–1381
 107. Kress-Gazit H, Wongpiromsarn T, Topcu U. 2011. Correct, reactive, high-level robot control. *IEEE Robotics and Automation Magazine* 18:65 – 74
 108. Wongpiromsarn T, Topcu U, Murray RM. 2012. Receding Horizon Temporal Logic Planning. *IEEE Trans. Automat. Contr.* 57:2817–2830
 109. Kozen D. 1983. Results on the Propositional mu-Calculus. *Theor. Comput. Sci.* 27:333–354
 110. Ehlers R. 2011. Generalized Rabin(1) Synthesis with Applications to Robust System Synthesis. In *{NASA} Formal Methods - Third International Symposium, {NFM} 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*
 111. Ehlers R, Raman V. 2016. Slugs: Extensible GR(1) Synthesis. In *Computer Aided Verification*
 112. Pnueli A, Sa'ar Y, Zuck LD. 2010. {JTLV}: {A} Framework for Developing Verification Algorithms. In *Computer Aided Verification, 22nd International Conference, {CAV} 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*
 113. Finucane C, Jing G, Kress-Gazit H. 2010. LTLMoP: Experimenting with Language, Temporal Logic and Robot Control. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2010)*. Taipei, Taiwan
 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} 2016, Buenos Aires, Argentina, September 19-22, 2016*
 115. Karaman S, Sanfelice RG, Frazzoli E. 2008. Optimal control of Mixed Logical Dynamical systems with Linear Temporal Logic specifications. In *Proceedings of the 47th IEEE Conference on Decision and Control, CDC 2008, December 9-11, 2008, Cancun, Mexico*
 116. Kwon Y, Agha G. 2008. {LTLC:} Linear Temporal Logic for Control. In *Hybrid Systems: Computation and Control, 11th International Workshop, {HSCC} 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*
 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} 2014, Hong Kong, China, May 31 - June 7, 2014*
 118. Lindemann L, Dimarogonas DV. 2017. Robust motion planning employing signal temporal logic. In *American Control Conference, {ACC} 2017, Seattle, WA, USA, May 24-26, 2017*
 119. Gol EA, Lazar M. 2013. Temporal logic model predictive control for discrete-time systems. In *Proceedings of the 16th international conference on Hybrid systems: computation and control, {HSCC} 2013, April 8-11, 2013, Philadelphia, PA, {USA}*
 120. Raman V, Donzé A, Sadigh D, Murray RM, Seshia SA. 2015. Reactive synthesis from signal

- temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems Computation and Control - HSCC '15*
121. Sadigh D, Kapoor A. 2016. Safe control under uncertainty with probabilistic signal temporal logic. In *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 - June 22, 2016*
 122. Jha S, Raman V. 2016. Automated synthesis of safe autonomous vehicle control under perception uncertainty. In *NASA Formal Methods - 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings*