Unmanned Systems, Vol. 0, No. 0 (2013) 1–19
© World Scientific Publishing Company

Synthesis of Control Protocols for Autonomous Systems

Tichakorn Wongpiromsarn[a], Ufuk Topcu[b], Richard M. Murray[c]

[a] *Ministry of Science and Technology, Bangkok, Thailand*
E-mail: tichakorn@gmail.com

[b] *University of Pennsylvania, Philadelphia, PA, USA*
E-mail: utopcu@seas.upenn.edu

[c] *California Institute of Technology, Pasadena, CA, USA*
E-mail: murray@cds.caltech.edu

This article provides a review of control protocol synthesis techniques that incorporate methodologies from formal methods and control theory to provide correctness guarantee for different types of autonomous systems, including those with discrete and continuous state space. The correctness of the system is defined with respect to a given specification expressed as a formula in linear temporal logic to precisely describe the desired properties of the system. The formalism presented in this article admits non-determinism, allowing uncertainties in the system to be captured. A particular emphasis is on alleviating some of the difficulties, e.g., heterogeneity in the underlying dynamics and computational complexity, that naturally arise in the construction of control protocols for autonomous systems.

Keywords: Autonomous systems; control architecture; linear temporal logic; receding horizon control

1. Introduction

Unmanned systems have the potential to improve man-power efficiencies and increase capabilities by augmenting or exceeding humans' capabilities. On the other hand, realization of this potential heavily relies on the autonomous functionalities these systems can deliver. Despite a number of proof-of-concept demonstrations, including the vehicles in the DARPA Urban Challenge [1] and Google driverless car [2], integration of autonomy has been slow. Take the status of the current generation of remotely-piloted aircraft as an example. The operator is still responsible for most of the functionalities, for example mission planning, generating and modifying tactics and contingency management. The functionality delegated to the vehicle is limited to straightforward, low-level tasks, for example point-to-point navigation or following pre-specified path segments.

One of the challenges in greater integration of autonomy is the lack of a suitable formalism for verifying system properties. Consequently, the design flow is often *ad hoc* and establishing trustworthiness of the systems is left to post-design simulations and testing. In particular, advances are needed in two directions: (i) Mathematically-based languages for unambiguously specifying the requirements the system needs to obey and properties of the environment in which the systems operates. The requirements and properties are often expressed as both high-level mission specifications and low-level constraints, e.g. actuation limitations. (ii) Automated methods to carry of the design of the control protocols specified in these languages. These design artifacts should either be amenable to post-design verification or correct (with respect to the formal specifications) by construction.

This article surveys recent progress in the specification and correct-by-construction synthesis of control protocols for autonomous systems to address the two needs above. The methods discussed in the article merge concepts from formal methods, including formal specification languages and discrete protocol synthesis, and those from controls, including optimization-based control and receding horizon implementations. A particular emphasis is on alleviating some of the difficulties, e.g., heterogeneity in the underlying dynamics and computational complexity, that naturally arise in the construction of autonomous protocols.

In the rest of the article, we discuss linear temporal logic [3–5] as a candidate specification language. We then formulate a protocol synthesis problem. The solution we present yields a hierarchical control structure for systems with either continuous or continuous and discrete dynamics. Each layer of the hierarchy uses a different model of the system and is responsible for ensuring the correctness of a different view of the constraints and specifications. The higher level uses a finite-state model of the system and is roughly responsible for the temporal logic specifications. Its construction relies on discrete protocol synthesis methods from formal methods [3,6,7]. The lower level uses a hybrid model (possibly with both continuous states and discrete modes) and is responsible for the correct implementation of the discrete directives issued by the higher level. The consistency between the two layers is ensured by the construction of a finite-state abstraction used by the higher-level. This construction is based on a series of finite-time, controlled reachability problems [8].

A third layer, which is inspired by the receding-horizon horizon control, aims to alleviate the computational complexity in reactive controller synthesis due to the possibly large number discrete states and branchings in the admissible environment behaviors. This layer uses a cruder abstraction and essentially decomposes the “long-horizon” reactive synthesis problem into “shorter-horizon” problems. Under certain conditions, the controllers from these short-horizon problems—when implemented in a receding horizon fashion—ensure the correctness with respect to the original, long-horizon specifications.

The material presented in the article builds on earlier research on formal methods [3,6,7] and recent work by the authors [8–12]. The rest of the article is organized as follows: Section 2 presents a case study of an autonomous system that motivates the developments discussed in this article. Section 3 summarizes formalisms used in this article for specifying systems and their requirements. Section 4.1 formulates a control protocol synthesis problem for autonomous systems. Section 4.2 presents a hierarchical control structure with two layers as a solution for this control protocol synthesis problem and describes correct-by-construction design for each layer as well as the construction of a finite-state abstraction of the physical system to ensure consistency between the two layers. Discrete protocol synthesis that enables correct-by-construction design of the higher layer is described in Section 5 and 6 for the case of deterministic and non-deterministic systems, respectively. Section 7 describes a method inspired by the receding-horizon horizon control to alleviate the computational complexity in reactive controller synthesis. A software toolbox implementing the approach presented in this article is described in Section 8. Finally, Section 9 presents the concluding remarks and open problems.

2. Motivating Example

Alice, an autonomous vehicle shown in Fig. 1, was built at the California Institute of Technology to compete in the 2007 DARPA Urban Challenge [1,13]. The competition required all the competing vehicles to navigate, in a fully autonomous manner, through a partially known urban-like environment populated with static and dynamic obstacles, including live traffic, while obeying traffic rules. In addition, the vehicles had to complete different tasks specified by a sequence of checkpoints that the vehicle had to cross. These tasks involved on- and off-road driving, parking, negotiating intersections and making U-turns. Hence, for the vehicles to successfully complete a given task, they needed to be capable of handling changes in their environment or operating condition (e.g. newly discovered obstacles) and reactively replanning in response to those changes (e.g. making a U-turn and finding a new route when the newly discovered obstacles fully block the road).

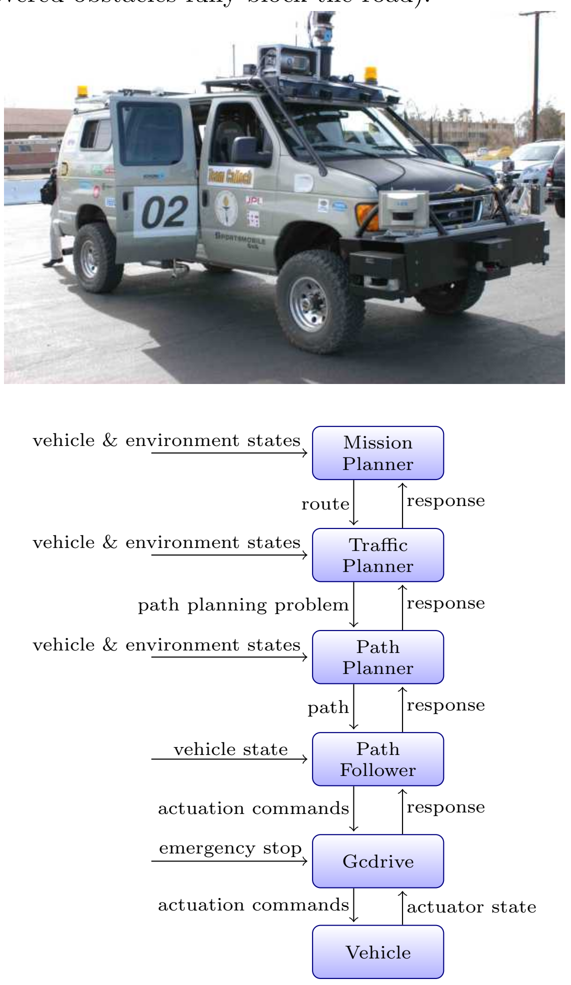


Fig. 1. *Top.* Alice, Team Caltech’s entry in the 2007 DARPA Urban Challenge. *Bottom.* Alice’s navigation protocol stack that reactively determines the motion of the vehicle based on the current state of its environment (as perceived by the sensing and estimation subsystems).

Alice was equipped with 25 CPUs and utilized a networked control system architecture to provide high performance and modular design. Its autonomous navigation relies on a protocol stack with the following software modules (Fig. 1) [13–15]:

- Mission Planner computes the route, i.e., a sequence of roads the vehicle has to navigate in order to cross a given sequence of checkpoints. It is also capable of re-computing the route when the response from Traffic Planner indicates that the previously computed route cannot be navigated successfully. This type of failure occurs, for example, when the road is blocked.

- Traffic Planner makes decisions to guide Alice at a high level. Specifically, based on the traffic rules and the current environment, it determines how Alice should navigate the route generated by Mission Planner, that is, whether it should stay in the travel lane or perform a passing maneuver, whether it should go or stop and whether it is allowed to reverse. In addition, it is responsible for intersection handling (e.g. keeping track of whether it is Alice's turn to go through an intersection). Based on these decisions, it sets up the constraints for the path planning problem.
- Path Planner generates a path that satisfies the directives and constraints determined by Traffic Planner.
- Path Follower computes control signals (e.g., acceleration and steering angle) such that the vehicle closely follows the path generated by Path Planner.
- Gcdrive is the low-level driving software for Alice. It contains logic to protect the physical hardware and only allows valid actuation commands computed by Path Follower to be executed by the actuators. Examples of the hardware protection logic include limiting the steering rate at low speeds and preventing shifting from occuring while the vehicle is moving. Furthermore, Gcdrive implements the emergency stop functionality for Alice and stops the vehicle when an externally-produced emergency stop command is received.

The particular protocol stack for Alice was designed and implemented completely by hand mostly in an *ad-hoc* manner. Although individual software modules were partially verified, the complete system was only validated through simulations and field tests. Detailed analysis after the competition, however, revealed an unforeseen interaction among the control modules and the physical environment that had not been witnessed in more than 300 miles of autonomous driving tests and thousands of hours of extensive simulations [16]. Such an unexpected interaction arose due to a mismatch in the abstraction of the physical system used at different levels of the hierarchy and under special circumstances that unfortunately included one of the tasks at the National Qualifying Event, led to unsafe behavior, causing Alice to get disqualified from the competition.

Alice exemplifies many autonomous systems, including other autonomous vehicles [1], in which navigation or control protocols are so complicated that only partial correctness guarantee can be provided through individual component analysis, simulations and field tests. In this article, we present formalisms and techniques that allow such protocols to be automatically designed with correctness guarantees. We employ formalisms from formal methods, including both modeling frameworks and specification languages to precisely describe the system and its correct behavior, respectively [3–5, 8]. With the precise description of both the system and its correct behavior, we describe techniques and tools from control theory and computer science that allow a control protocol that guarantees the correct behavior of the system to be automatically designed.

3. Preliminaries

In this article, we use the formalisms from formal methods as explained in [3] to describe systems and their correct behavior. We summarize those formalisms in this section and refer the reader to [3] for more details. Given a set X, let X^*, X^ω and X^+ denote the set of finite, infinite and nonempty finite strings, respectively, of X and let $|X|$ denote the cardinality of X. For sequences π, π_1 and π_2, let $\pi_1\pi_2$ denote a sequence obtained by concatenating π_1 and π_2 and let π^ω denote an infinite sequence obtained by concatenating π infinitely many times.

3.1. *Transition Systems*

A transition system is a mathematical description of the behavior of systems with discrete inputs, outputs, internal states and transitions between the states. Its behavior is formalized by *atomic propositions* that express important characteristics of individual states of the system. Roughly, a *proposition* is a statement that can be either true or false, but not both. An *atomic proposition* is a proposition whose truth or falsity does not depend on the truth or falsity of any other proposition. For example, a statement "Traffic light is green" is an atomic proposition whereas a statement "Traffic light is either green or red" is not an atomic proposition.

Definition 3.1. A *transition system* TS is a tuple $TS = (S, Act, \rightarrow, I, AP, L)$, where

- S is a set of states,
- Act is a set of actions,
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation,
- $I \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions and
- $L : S \rightarrow 2^{AP}$ is a labeling function.

We use the relation notation, $s \xrightarrow{\alpha} s'$, to denote $(s, \alpha, s') \in \rightarrow$. TS is called *finite* if S, Act and AP are finite.

Complex systems are typically composed of multiple components that can be executed at the same time. Such systems are referred to as "concurrent systems". Suppose each component of the system can be modeled by a transition system. Under the assumption of synchronous operation of all the components, the complete system can be constructed based on the composition (by hand-shaking) of the transition systems representing individual components as follows.

Definition 3.2. Let $TS_1 = (S_1, Act_1, \rightarrow_1, I_1, AP_1, L_1)$ and $TS_2 = (S_2, Act_2, \rightarrow_2, I_2, AP_2, L_2)$ be transition systems. Their parallel composition, $TS_1 || TS_2$ is the transition system defined by

$$TS_1||TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L),$$

where $L(\langle s_1, s_2\rangle) = L_1(s_1) \cup L_2(s_2)$ and $\rightarrow$ is defined by the following rules:

- If $\alpha \in Act_1 \cap Act_2$, $s_1 \xrightarrow{\alpha} s_1'$ and $s_2 \xrightarrow{\alpha} s_2'$, then $\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1', s_2'\rangle$.
- If $\alpha \in Act_1 \setminus Act_2$ and $s_1 \xrightarrow{\alpha} s_1'$, then $\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1', s_2\rangle$.
- If $\alpha \in Act_2 \cap Act_1$ and $s_2 \xrightarrow{\alpha} s_2'$, then $\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1, s_2'\rangle$.

Example 1. Consider a system of traffic lights shown in Fig. 2. For $i \in \{1, 2\}$, we let g_i denote an atomic proposition stating that light T_i is green. Then, $T_1 = (S_1, Act_1, \rightarrow_1, I_1, AP_1, L_1)$ is a finite transition system where $S_1 = \{s_{1,1}, s_{1,2}\}$, $Act_1 = \{\alpha_1\}$, $\rightarrow_1 = \{(s_{1,1}, \alpha_1, s_{1,2}), (s_{1,2}, \alpha_1, s_{1,1})\}$, $I_1 = \{s_{1,1}\}$, $AP_1 = \{g_1\}$ and $L : S_1 \rightarrow 2^{AP_1}$ is defined by $L(s_{1,1}) = \emptyset$ and $L(s_{1,2}) = \{g_1\}$. Also, $T_2 = (S_2, Act_2, \rightarrow_2, I_2, AP_2, L_2)$ is a finite transition system where $S_2 = \{s_{2,1}, s_{2,2}\}$, $Act_2 = \{\alpha_2\}$, $\rightarrow_2 = \{(s_{2,1}, \alpha_2, s_{2,2}), (s_{2,2}, \alpha_2, s_{2,1})\}$, $I_2 = \{s_{2,1}\}$, $AP_2 = \{g_2\}$ and $L : S_2 \rightarrow 2^{AP_2}$ is defined by $L(s_{2,1}) = \emptyset$ and $L(s_{1,2}) = \{g_2\}$. Fig. 3 shows the graphical representation of T_1, T_2 and their parallel composition $T_1||T_2$.

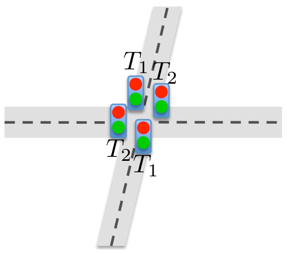


Fig. 2. A system of traffic lights considered in Example 1.

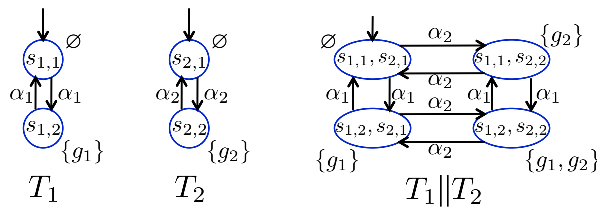


Fig. 3. The transition systems representing the models of traffic lights in Example 1.

Given a transition system $TS = (S, Act, \rightarrow, I, AP, L)$, $s \in S$ and $\alpha \in Act$, we let $Act(s) = \{\alpha \in Act : \exists s' \in S$ such that $s \xrightarrow{\alpha} s'\}$ denote the set of enabled actions in s, $Post(s, \alpha) = \{s' \in S : s \xrightarrow{\alpha} s'\}$ and $Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha)$ denote the set of direct successors of s. We say that TS is *action-deterministic* if and only if $|I| \leq 1$ and $|Post(s, \alpha)| \leq 1$ for all $s \in S$ and $\alpha \in Act$. A sequence of states, either finite $\pi = s_0 s_1 \ldots s_n$, or infinite $\pi = s_0 s_1 \ldots$, is a *path fragment* if $s_{i+1} \in Post(s_i)$ for all $i \geq 0$. A *path* is a path fragment such that $s_0 \in I$ and it is either a finite path fragment that ends in a state s with $Post(s) = \emptyset$ or an infinite path fragment. We denote the set of paths in TS by $Path(TS)$. The *trace* of an infinite path fragment $\pi = s_0 s_1 \ldots$ is defined by $trace(\pi) = L(s_0)L(s_1)\ldots$. The set of traces of TS is defined by $Trace(TS) = \{trace(\pi) : \pi \in Path(TS)\}$.

Remark 3.3. Transition systems that are not action-deterministic are those in which some action, when applied in some state, leads to several possible next states. Hence, they can be used to capture uncertainties in the system, especially those arise from difference choices of valid environment behavior over which the system does not have control.

3.2. *Linear Temporal Logic*

Linear temporal logic (LTL) is a formal language for describing linear-time properties. Its use as a specification language was introduced by Pnueli in the 1970s [17]. Since then, LTL has been demonstrated to be an appropriate specification language for reasoning about various kinds of systems, including programs and resource allocators [18], artificial intelligence [19] and discrete event systems [20,21].

An LTL formula is built up from a set of atomic propositions and two kinds of operators: logical connectives and temporal modal operators. The logic connectives are those used in propositional logic: *negation* ($\neg$), *disjunction* ($\vee$), *conjunction* ($\wedge$) and *material implication* ($\implies$). The temporal modal operators include *next* ($\bigcirc$), *always* ($\Box$), *eventually* ($\Diamond$) and *until* ($\mathcal{U}$). Specifically, an LTL formula over a set AP of atomic propositions is inductively defined as follows:

(1) *True* is an LTL formula,
(2) any atomic proposition $p \in AP$ is an LTL formula and
(3) given LTL formulas φ, φ_1 and φ_2, $\neg\varphi$, $\varphi_1 \vee \varphi_2$, $\bigcirc\varphi$ and $\varphi_1 \mathcal{U} \varphi_2$ are also LTL formulas.

Additional operators can be derived from the logical connectives $\vee$ and $\neg$ and the temporal modal operator $\mathcal{U}$. For example, $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \implies \varphi_2 = \neg\varphi_1 \vee \varphi_2$, $\Diamond\varphi = True\ \mathcal{U}\ \varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$.

LTL formulas are interpreted on infinite strings $\sigma = \sigma_0\sigma_1\sigma_2\ldots$ where $\sigma_i \in 2^{AP}$ for all $i \geq 0$. Such infinite strings are referred to as *words*. The satisfaction relation is denoted by $\models$, i.e., for a word σ and an LTL formula φ, we write $\sigma \models \varphi$ if and only if σ satisfies φ. The satisfaction relation is defined inductively as follows:

- $\sigma \models True$,
- for an atomic proposition $p \in AP$, $\sigma \models p$ if and only if $p \in \sigma_0$,
- $\sigma \models \neg\varphi$ if and only if $\sigma \not\models \varphi$,
- $\sigma \models \varphi_1 \wedge \varphi_2$ if and only if $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$,
- $\sigma \models \bigcirc\varphi$ if and only if $\sigma_1\sigma_2\ldots \models \varphi$ and

- $\sigma \models \varphi_1 \,\mathcal{U}\, \varphi_2$ if and only if there exists $j \geq 0$ such that $\sigma_j\sigma_{j+1}\ldots \models \varphi_2$ and for all i such all $0 \leq i < j$, $\sigma_i\sigma_{i+1}\ldots \models \varphi_1$.

Let φ be an LTL formula over AP. The linear-time property induced by φ is defined as $Words(\varphi) = \{\sigma \in (2^{AP})^\omega : \sigma \models \varphi\}$. Given a transition system TS, its infinite path fragment π and an LTL formula φ over AP, we say that π satisfies φ, denoted $\pi \models \varphi$, if $trace(\pi) \models \varphi$. Finally, we say that TS satisfies φ, denoted $TS \models \varphi$, if $Trace(TS) \subseteq Words(\varphi)$.

Given a proposition p, examples of widely used LTL formulas include a safety formula of the form $\Box p$ (read as "always p") and a reachability formula of the form $\Diamond p$ (read as "eventually p"). A word satisfies $\Box p$ if p remains invariantly true at all positions of the word whereas it satisfies $\Diamond p$ if p becomes true at least once in the word. By combining the temporal operators, we can express more complex properties. For example $\Box\Diamond p$ states that p holds infinitely often in the word.

Example 2. Consider the system of traffic lights described in Example 1. Its desired properties might include:

- "At least one of the lights is always on" is a safety property and can be expressed in LTL as $\Box(g_1 \vee g_2)$.
- "Two lights are never green at the same time" is also a safety property and can be expressed in LTL as $\Box(\neg g_1 \vee \neg g_2)$.
- "T_1 will turn green infinitely often" is known as a "progress" property and can be expressed in LTL as $\Box\Diamond g_1$.

Example 3. As described in Section 2, an autonomous vehicle competing in the DARPA Urban Challenge is required to follow traffic rules as well as completing a task specified by a sequence of checkpoints that the vehicle has to cross. This autonomous driving problem is simplified in Fig. 4. Examples of some desired properties and LTL formulas expressing those properties are given below.

Traffic rules:

- No collision: $\Box\big(dist(x, Obs) \geq X_{safe} \wedge dist(x, Loc(Veh)) \geq X_{safe}\big)$ where $dist$ is a distance function, Obs and $Loc(Veh)$ represent the positions of the nearest obstacle and vehicle, respectively, and X_{safe} is a pre-specified parameter for the required safety distance.
- Obey speed limits: $\Box\big((x \in \text{Reduced\_Speed\_Zone}) \implies (v \leq v_{reduced})\big)$ where $v_{reduced}$ is a pre-specified parameter for the maximum speed in Reduced_Speed_Zone.

Goals:

- Eventually visit the checkpoint: $\Diamond(x = \text{ck\_pt})$ where ck_pt denote the position of the checkpoint.

Note that, $dist(x, Obs) \geq X_{safe}$, $dist(x, Loc(Veh)) \geq X_{safe}$, $x \in \text{Reduced\_Speed\_Zone}$ and $v \leq v_{reduced}$ are considered to be atomic propositions in this example as they are evaluated to either true or false, given the state (x, v, Obs and $Loc(Veh)$) of the system.

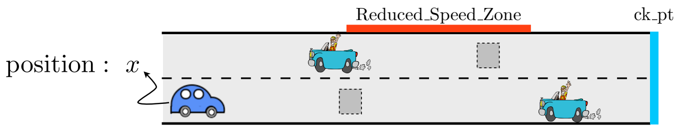


Fig. 4. A simplified autonomous driving problem considered in Example 3.

Remark 3.4. Properties typically studied in the control and hybrid systems domains are safety (usually in the form of constraints on the system state) and stability (i.e., convergence to an equilibrium or a desired state). However, these properties are not rich enough to describe desired properties of many autonomous systems. For example, desired properties of an autonomous vehicle such as staying in the travel lane unless there is an obstacle blocking the lane and visiting a certain area infinitely often cannot be expressed in terms of safety and stability only. The expressiveness of LTL allows such properties to be specified.

Timed extensions of temporal logics include metric temporal logic [22] and timed computation tree logic [3], allowing timing requirements to be incorporated. Probabilistic temporal logics such as computation tree logic have also been introduced to capture the probabilistic aspect and allow quantitative reasoning of systems with uncertainties [3].

3.3. *Automata*

Finite state automata provide another representation of linear-time properties that can be handled computationally. Furthermore, as we will see later, there is a tight relationship between LTL and finite state automata that will be exploited in control protocol synthesis.

Definition 3.5. A *non-deterministic Buchi automaton* (NBA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ where

- Q is a finite set of states,
- Σ is a finite set, called an alphabet,
- $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation,
- $Q_0 \subseteq Q$ is a set of initial states and
- $F \subseteq Q$ is a set of accepting (or final) states.

We use the relation notation, $q \xrightarrow{\sigma} q'$, to denote $(q, \sigma, q') \in \delta$.

Consider an infinite string $\sigma = \sigma_0\sigma_1 \ldots \in \Sigma^\omega$. A *run* for σ in an NBA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ is an infinite sequence of states $q_0q_1\ldots$ such that $q_0 \in Q_0$ and $q_i \xrightarrow{\sigma_i} q_{i+1}$ for all $i \geq 0$. A run is *accepting* if there exist infinitely many $j \geq 0$ such that $q_j \in F$. A string $\sigma \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there is an accepting run of σ in $\mathcal{A}$. The *language* accepted by

$\mathcal{A}$, denoted by $\mathcal{L}_\omega(\mathcal{A})$, is the set of all accepted strings of $\mathcal{A}$. A set of infinite strings $\mathcal{L}_\omega \subseteq \Sigma^\omega$ is called an *ω-regular language* if there is an NBA $\mathcal{A}$ such that $\mathcal{L}_\omega = \mathcal{L}_\omega(\mathcal{A})$.

It can be shown that for any LTL formula φ over AP, there exists an NBA $\mathcal{A}_\varphi$ with alphabet $\Sigma = 2^{AP}$ that accepts all words and only those words over AP that satisfy φ, i.e., $\mathcal{L}_\omega(\mathcal{A}_\varphi) = Words(\varphi) = \{\sigma \in (2^{AP})^\omega : \sigma \models \varphi\}$ [3]. Such $\mathcal{A}_\varphi$ can be automatically constructed using existing tools [23] with the worst-case complexity that is exponential in the length of φ.

Example 4. An NBA that recognizes (i.e., accepts all and only words satisfying) each of the desired properties of the traffic light system in Example 2 is shown in Fig. 5. For example, an accepting run of the automaton in Fig. 5(a), needs to visit state q_0 infinitely often. Hence, either g_1 or g_2 has to be included at all the positions in a string accepted by this automaton. As a result, this automaton accepts all the only words satisfying an LTL formula $\Box(g_1 \vee g_2)$.

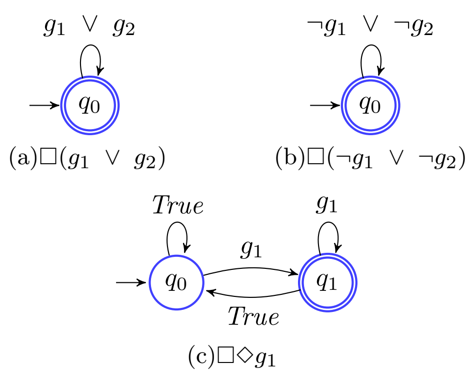


Fig. 5. Non-deterministic Buchi automata that recognize the desired properties of the traffic light system in Example 2. An arrow without a source points to an initial state. Accepting states are drawn with a double circle.

3.4. *Control Protocols*

In this article, we are interested in synthesizing a control protocol for a transition system to ensure that a given LTL specification is satisfied. We define a control protocol for a transition system as follows.

Definition 3.6. Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system. A *control protocol* for TS is a function $\mathcal{C} : S^+ \rightarrow Act$ such that $\mathcal{C}(s_0 s_1 \ldots s_n) \in Act(s_n)$ for all $s_0 s_1 \ldots s_n \in S^+$.

A control protocol $\mathcal{C}$ for a transition system TS essentially restricts the non-deterministic choices in TS by picking an action based on the path fragment that leads to the current state of the system, leaving only non-deterministic choices that arise from the factors over which the system does not have control. Hence, $\mathcal{C}$ induces a transition system $TS^{\mathcal{C}}$ that formalizes the behavior of TS under control protocol $\mathcal{C}$. In general, $TS^{\mathcal{C}}$ contains all the states in S^+ and hence may not be finite even though TS is finite. However, for special cases where $\mathcal{C}$ is a memoryless or a finite memory control protocol, it can be shown that $TS^{\mathcal{C}}$ can be identified with a finite transition system. Roughly, a memoryless control protocol always picks the action based only on the current state of TS, regardless of the path fragment that led to that state. A finite memory control protocol also maintains its "mode" and picks the action based on its current mode and the current state of TS.

Example 5. Consider the transition system $T_1||T_2 = (S, Act, \rightarrow, I, AP, L)$ that represents the complete traffic light system in Example 1 (see Fig. 3). Define a control protocol $\mathcal{C} : S^+ \rightarrow Act$ such that

- $\mathcal{C}(\langle s_{1,1}s_{2,1}\rangle) = \alpha_1$,
- $\mathcal{C}(\pi\langle s_{1,1}s_{2,2}\rangle\langle s_{1,1}s_{2,1}\rangle) = \alpha_1$,
- $\mathcal{C}(\pi\langle s_{1,2}s_{2,1}\rangle\langle s_{1,1}s_{2,1}\rangle) = \alpha_2$,
- $\mathcal{C}(\pi\langle s_{1,1}s_{2,1}\rangle\langle s_{1,2}s_{2,1}\rangle) = \alpha_1$ and
- $\mathcal{C}(\pi\langle s_{1,1}s_{2,1}\rangle\langle s_{1,1}s_{2,2}\rangle) = \alpha_2$

for any $\pi \in S^*$. The transition systems induced by $\mathcal{C}$ is shown in Fig. 6. It can be easily checked that this transition system satisfies $\Box(\neg g_1 \vee \neg g_2)$ and $\Box\Diamond g_1$. However, it violates $\Box(g_1 \vee g_2)$.

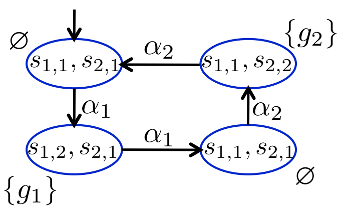


Fig. 6. $(T_1||T_2)^{\mathcal{C}}$, the transition systems induced by applying $\mathcal{C}$ defined in Example 5 on the traffic light system $T_1||T_2$.

4. Control Protocol Synthesis

4.1. *Problem Formulation*

We consider systems that may be deterministic (e.g., in the case of closed systems whose behaviors are generated purely by the system itself without any external influence) or non-deterministic (e.g., in the case of open systems whose behaviors can be affected by external influence). Non-determinism can be used to capture uncertainties in the system, especially those arise from difference choices of valid environment behavior over which the system does not have control. We assume that at any time instance, the state of the system can be precisely observed.

Depending on the type of the system, its state space may be discrete, continuous or contains both the discrete and continuous components. For a purely discrete system, we assume that it can be modeled as a finite transition system. For a purely continuous system, we assume that its state evolves based on the continuous-time dynamics

$$\dot{\xi}(t) = f(\xi(t), d(t), u(t)), \qquad \xi(t) \in \mathcal{X}, d(t) \in \mathcal{D}, u(t) \in \mathcal{U}, \tag{1}$$

or the discrete-time dynamics

$$\xi[k+1] = f(\xi[k], d[k], u[k]), \qquad \xi[k] \in \mathcal{X}, d[k] \in \mathcal{D}, u[k] \in \mathcal{U}, \tag{2}$$

where $\mathcal{X} \subset \mathbb{R}^n$ is the continuous state space, $\mathcal{D} \subset \mathbb{R}^p$ is the set of exogenous disturbances and $\mathcal{U} \subset \mathbb{R}^m$ is the set of admissible control input. Finally, a hybrid system framework [24] is used to modeled a system with both discrete-state and continuous-state components.

Control Protocol Synthesis Problem: Given a continuous, discrete or hybrid model of the dynamics of a system and a specification expressed as an LTL formula φ, automatically synthesize a control protocol for the system to satisfy φ.

We say that the system is *correct* if the specification φ is satisfied. Note that for a non-deterministic system, its correctness needs to be interpreted with respect to the non-deterministic choices over which the system does not have control. In this case, we require the control protocol to ensure that the specification φ is satisfied for all the possible non-deterministic choices (e.g., all the possible behavior of the environment).

Example 6. Consider the system of traffic lights described in Example 1. The system is modeled as $T_1||T_2$ whose state space is discrete. Since $T_1||T_2$, is action-deterministic, the system, in this case, is deterministic.

Example 7. Consider the robot motion planning problem where the robot navigates an area that is partitioned into cells as shown in Fig. 7. The dynamics of the robot is abstracted to a finite transition system TS shown on the right of Fig. 7. The state of TS represents the cell occupied by the robot. If TS is action-deterministic (i.e., each action, when applied in each state, uniquely determines the next state), then the system is deterministic. Otherwise, the system is non-deterministic. In this case, non-determinism potentially arises due to disturbances that affect the dynamics of the robot, leading to multiple possible next states when some action is applied in some state. The desired property of the system is for a robot to visit cell C_8, then C_1 and then cover C_{10}, C_{17} and C_{25} in any order while always avoiding cells C_2, C_{14} and C_{18}. This property can be expressed in LTL as $\diamond\big(C_8 \wedge \diamond(C_1 \wedge \diamond C_{10} \wedge \diamond C_{17} \wedge \diamond C_{25})\big) \wedge \square\neg(C_2 \vee C_{14} \vee C_{18})$.

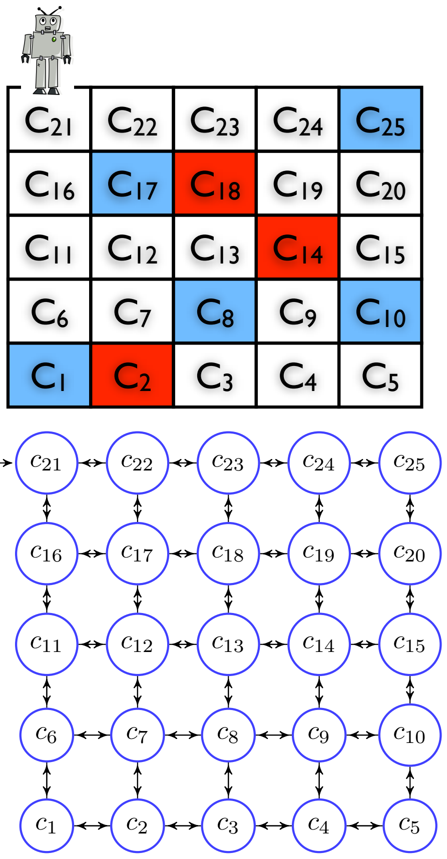


Fig. 7. The robot motion planning problem in Example 7 with the grid-based world model.

Example 8. Consider the simplified autonomous driving problem described in Example 3. The system consists of the autonomous vehicle, obstacles and other vehicles. If the obstacles and other vehicles are not stationary and their motion is not known exactly, then the system is non-deterministic since the system does not have control over the motion of the obstacles and other vehicles. In this case, a control protocol for this system needs to ensure that the desired properties described in Example 3 are satisfied for all the possible motion (i.e., behavior) of the obstacles and other vehicles.

4.2. *Abstraction-Based Approach*

For discrete systems that contain a finite number of states, existing results from formal methods can be employed to synthesize a control protocol to ensure that specification φ is satisfied. In the case where the system is deterministic, the control protocol synthesis problem can be formulated as a satisfiability problem, commonly known as a model checking problem [3]. We say that φ is *satisfiable* if the system can satisfy φ. We further discuss this approach in Section 5. In the case where the system is non-deterministic, the control protocol synthesis problem can be treated as a two-player game between the system and the environment (i.e., adversary): the environment attempts to falsify φ while the system attempts to satisfy φ. Such "reactive module synthesis" has been considered in [6,7]. We say that φ is *realizable* if there exists a control protocol that ensures that φ is satisfied no matter what the environment does. We further discuss this approach in Section 6.

For the case where the system is modeled as a continuous or hybrid system, the number of states is infinite; hence, discrete control protocol synthesis, including both model-checking-based and reactive module synthesis, cannot be directly employed. In this case, a common approach to the control protocol synthesis problem is to construct a finite transition system $TS = (S, Act, \to, I, AP, L)$ that serves as an abstract model of the system [8, 25–30]. Discrete control protocol synthesis as previously mentioned can then be employed to synthesize a (discrete) control protocol for this finite transition system to ensure that the specification is satisfied. This leads to a hierarchical, two-layer design as shown in Fig. 8 with the following layers:

(a) a discrete planner computes a discrete plan satisfying φ based on the abstract, finite state model and
(b) a continuous controller computes continuous control signal for the system to implement the discrete plan based on the physical, infinite state model.

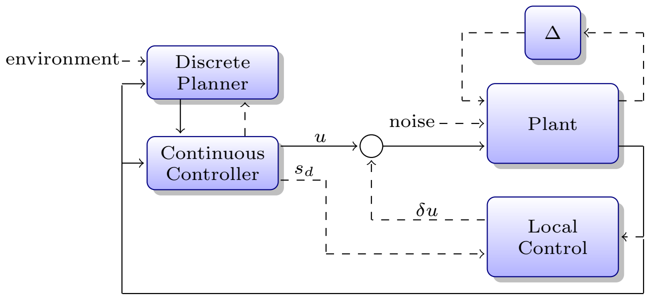


Fig. 8. The hierarchical control implementation of the abstraction-based approach. Besides the components discussed in this article, Δ, which captures uncertainties in the plant model, may be added to make the model more realistic. Additionally, a local control may be implemented to account for the effect of noise, disturbances and unmodeled dynamics. The inputs and outputs of these two components, not considered in this paper, are drawn in dashed.

In this hierarchical control design, disturbances that affect the dynamics of the system may be handled at either the discrete planner or the continuous controller level. The correctness of this abstraction-based approach relies on the correctness of the abstraction of an infinite-state system into a finite state model, which has to ensure that the continuous controller can *implement* or *simulate* any plan generated by the discrete planner. Roughly, this requires that the continuous execution (e.g., the sequence of states of the physical system) is equivalent to the plan (e.g., the sequence of abstract states) generated by the discrete planner. We refer the reader to [24, 31] for the exact definition of language equivalence and simulation. It can be shown that the continuous execution is guaranteed to preserve the correctness of the discrete plan, provided that the abstraction is correct and φ is *stutter invariant* [24]. Specifically, a specification φ is stutter invariant if for any word $\sigma = \sigma_0^{n_0}\sigma_1^{n_1}\sigma_2^{n_2}\ldots$ with $\sigma \models \varphi$ where $n_0, n_1, n_2, \ldots$ are natural numbers, $\sigma_0^{m_0}\sigma_1^{m_1}\sigma_2^{m_2}\ldots \models \varphi$ for any natural numbers $m_0, m_1, m_2, \ldots$ [32].

There are two main abstraction techniques: fixed abstraction and sampling-based approaches. Roughly, the fixed abstraction approaches (Fig. 9) work by partitioning the continuous state space into a finite number of cells, each of which serves as a state of TS. To ensure the correctness of the abstraction, the partition needs to be *proposition preserving*, i.e., for any atomic proposition $p \in AP$ and any states ξ_1 and ξ_2 that belong to the same cell in the partition, ξ_1 satisfies p if and only if ξ_2 also satisfies p. In addition, the transition relation of TS is defined by "reachability" between cells, i.e., $s \xrightarrow{\alpha} s'$ for some $\alpha \in Act$ only if from any point ξ in the cell represented by s, there exists a continuous control signal that takes the system from point ξ to a point in the cell represented by s'. If the disturbance is to be handled by the continuous controller, the control signal needs to ensure that s' is reached for any allowable disturbance.

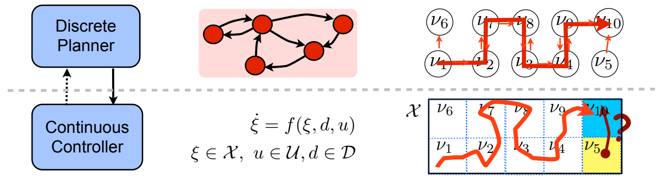


Fig. 9. The fixed abstraction approach where the continuous state space is partitioned into cells. These cells then serve as the states of the transition system that represents the abstract model of the system. This abstract model is then used by the discrete planner to generate a discrete plan satisfying a given specification.

Fixed Abstraction: Several fixed abstraction methods have been proposed for different type of dynamics. For example, a continuous-time, time-invariant model was considered in [29], [26] and [27] for special cases of fully actuated ($\dot{s} = u$), kinematic ($\dot{s} = A(s)u$) and piecewise affine dynamics, respectively. A discrete-time, time-invariant model was considered in [8] and [25] for special cases of piecewise affine and controllable linear systems, respectively. Reference [28] deals with more general dynamics by relaxing the bisimulation requirement and using the notion of approximate simulation [33, 34]. Except in special cases, fixed abstraction approaches typically cannot provide completeness guarantees, i.e., they typically generate TS that is an under-approximation of the physical system; hence, the specification may not be satisfiable or realizable based on TS even though it is for the physical system.

For completeness, we provide details for the technique presented in [8]. Consider the case where the state of the system evolves based on the discrete-time linear, time-invariant, state-space model

$$\xi[t+1] = A\xi[t] + B_u u[t] + B_d w[t], \tag{3}$$

where for all t, $\xi[t] \in \mathcal{X}$ is the state of the system, $u[t] \in \mathcal{U}$ is the control input, $d[t] \in \mathcal{D}$ is the exogenous disturbance and $\xi[0] \in \mathcal{X}$. To simplify the reachability verification, we consider the restricted case where $\mathcal{X} \subseteq \mathbb{R}^n$, $\mathcal{U} \subseteq \mathbb{R}^m$ and $\mathcal{D}$ are polyhedral sets. Furthermore, we assume that each atomic proposition $p \in AP$ is given as a linear inequality that essentially defines a halfspace of $\mathbb{R}^n$.

The first step of the abstraction procedure is to construct a proposition preserving, polytopic partition $\{\mathcal{X}_0, \dots, \mathcal{X}_P\}$ of $\mathcal{X}$ based on the set AP of atomic propositions. (Since each $p \in AP$ defines a halfspace of $\mathcal{X}$, a proposition preserving partition $\{\mathcal{X}_0, \dots, \mathcal{X}_P\}$ of $\mathcal{X}$ can be constructed such that each cell $\mathcal{X}_i, i \in \{0, \dots, P\}$ is a polytope.) Define a finite transition system $TS = (S, Act, \rightarrow, I, AP, L)$ where $S = \{\mathcal{X}_0, \dots, \mathcal{X}_P\}$, $Act = \{\alpha_0, \dots, \alpha_P\}$, $I = \{\mathcal{X}_I\}$ such that $\xi[0] \in \mathcal{X}_I$ and L maps each cell $\mathcal{X}_i$ to the set of propositions satisfied by all the states in $\mathcal{X}_i$. The transition relation $\rightarrow$ can be defined based on the notion of finite-time reachability as follows.

Consider arbitrary $i, j \in \{0, \dots, P\}$. We say that $\mathcal{X}_j$ is *finite-time reachable* from $\mathcal{X}_i$ only if starting from any $\xi[0] \in \mathcal{X}_i$, there exists a finite horizon length $N \in \{0, 1, \dots\}$ such that for any allowable disturbance, there exists a sequence of admissible control inputs $u[0], u[1], \dots, u[N-1] \in \mathcal{U}$ that take the system to a point in $\mathcal{X}_j$ without leaving $\mathcal{X}_i \cup \mathcal{X}_j$, i.e.,

$$\xi[N] \in \mathcal{X}_j, \text{ and} \tag{4}$$

$$\xi[t] \in \mathcal{X}_i \cup \mathcal{X}_j, \text{ for all } t \in \{0, \dots N\}. \tag{5}$$

To verify this reachability relation, we compute the set $S_0^{i,j}$ of states, starting from which conditions (4) and (5) can be satisfied under the system dynamics (3) for a pre-specified horizon length N. If $\mathcal{X}_i \subseteq S_0^{i,j}$, we can conclude that $\mathcal{X}_j$ is finite-time reachable from $\mathcal{X}_i$ and add the transition $\mathcal{X}_i \xrightarrow{\alpha_j} \mathcal{X}_j$ to TS.

Exploiting the assumption that $\mathcal{U}$, $\mathcal{D}$, $\mathcal{X}_i$ and $\mathcal{X}_j$ are polyhedral sets and hence can be specified as a set of linear inequalities, it can be shown [8] that if $\mathcal{D}$ is closed and bounded, then $S_0^{i,j}$ can be obtained by computing the projection of a certain polytope on $\mathbb{R}^n$. Specifically,

$$S_0^{i,j} = \Big\{ s \in \mathbb{R}^n : \exists \hat{u} \in \mathbb{R}^{mN} \text{ such that } L \begin{bmatrix} s \\ \hat{u} \end{bmatrix} \leq M - G\hat{d}, \\ \forall \hat{d} \in \bar{\mathcal{D}}^N \Big\}, \tag{6}$$

where the matrices L, M and G can be constructed from A, B_u, B_d and the description of $\mathcal{U}$, $\mathcal{X}_i$ and $\mathcal{X}_j$.

The computation of $S_0^{i,j}$ above not only allows us to establish the reachability relation, but it also allows us to refine the partition of $\mathcal{X}$ to partially alleviate the conservatism (due to the constraint on the control input as well as a specific choice of the finite horizon N) of this fixed abstraction approach by increasing the number of valid transitions of TS. The underlying idea is to split $\mathcal{X}_i$ into $\mathcal{X}_i \cap S_0^{i,j}$ and $\mathcal{X}_i \setminus S_0^{i,j}$. This partition refinement procedure can be repeated until some pre-specified termination criteria such as the lower bound on the volume of each cell, are met. At termination, it generates a finite transition system whose states correspond to the cells in the refined partition of the continuous state space. This finite transition system serves as an abstract model of the physical system to be used by the discrete planner as shown in Fig. 9.

Example 9. As shown in [35], the dynamics of a point-mass omnidirectional vehicle is given by

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} \dot{x} \\ \dot{y} \\ \frac{2mL^2}{J}\dot{\theta} \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \\ u_\theta \end{bmatrix}, \tag{7}$$

with the following constraints on the control efforts:

$$u_x^2(t) + u_y^2(t) \leq \left(\frac{3 - |u_\theta(t)|}{2}\right)^2 \text{ and } |u_\theta(t)| \leq 3. \tag{8}$$

To decouple the constraints (8), we set $|u_x(t)| \leq \sqrt{0.5}$, $|u_y(t)| \leq \sqrt{0.5}$ and $|u_\theta(t)| \leq 1$ for all t. As we will see later, these conservative bounds allow us to simplify the reachability problem.

Suppose we are only interested in the translational (x and y) components of the vehicle state. Discretizing the dynamics (7) with time step 0.1, we obtain the following discrete-time linear, time-invariant, state-space model

$$\begin{bmatrix} z[t+1] \\ v_z[t+1] \end{bmatrix} = \begin{bmatrix} 1 & 0.0952 \\ 0 & 0.9048 \end{bmatrix} \begin{bmatrix} z[t] \\ v_z[t] \end{bmatrix} + \begin{bmatrix} 0.0048 \\ 0.0952 \end{bmatrix} u_z \tag{9}$$

where z represents either x or y and v_z represents the rate of change in z. Let $\mathcal{X}_z$ be the domain of the vehicle state projected onto the (z, v_z) coordinates. We restrict the domain $\mathcal{X}_z$ to $[z_{min}, z_{max}] \times [-1, 1]$ and partition $\mathcal{X}_z$ as

$$\mathcal{X}_z = \bigcup_{i \in \{z_{min}+1, \dots, z_{max}\}} \mathcal{X}_{z,i}^- \cup \bigcup_{i \in \{z_{min}+1, \dots, z_{max}\}} \mathcal{X}_{z,i}^+,$$

where $\mathcal{X}_{z,i}^- = [i-1, i] \times [-1, 0]$ and $\mathcal{X}_{z,i}^+ = [i-1, i] \times [0, 1]$ as shown in Fig. 10.

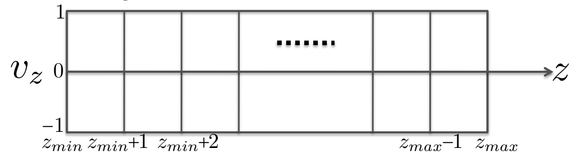


Fig. 10. The proposition preserving partition of the domain C_z

Since the dynamics and the constraints on the control efforts for the x and y components of the vehicle state are decoupled, we apply the reachability verification and partition refinement technique described earlier for the x

and y components separately for the sake of computational efficiency. Furthermore, since the vehicle dynamics (7) are translationally invariant, we can use a similar partition for all $\mathcal{X}^+_{z,i}$ and a similar partition for all $\mathcal{X}^-_{z,i}$, $i \in \{z_{min}+1, \ldots, z_{max}\}$. Exploiting these symmetries allow us to simplify the abstraction procedure by considering only 4 adjacent cells $\mathcal{X}^+_{z,i}$, $\mathcal{X}^-_{z,i}$, $\mathcal{X}^+_{z,i+1}$ and $\mathcal{X}^-_{z,i+1}$ for some $i \in \{z_{min}+1, \ldots, z_{max}-1\}$. Other cells can then be partitioned with the reachability relation defined as in these 4 cells.

The partition refinement technique based on the computation of $S_0^{i,j}$ with horizon length $N = 10$ and the lower bound $Vol_{min} = 0.1$ on the volume of each cell generates a partition of $\mathcal{X}^+_{z,i}$ with 5 cells and a partition of $\mathcal{X}^-_{z_i}$ with 6 cells for each $i \in \{z_{min}+1, \ldots, z_{max}\}$ as shown in Fig. 11.

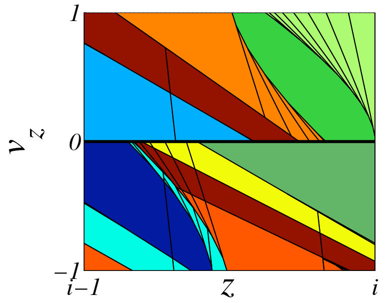


Fig. 11. The partition of cells $\mathcal{X}^+_{z,i}$ and $\mathcal{X}^-_{z,i}$ where $i \in \{z_{min}+1, \ldots, z_{max}\}$ obtained from the partition refinement technique with horizon length $N = 10$.

Sampling-Based Abstraction: A sampling-based method has been proposed for deterministic systems with discrete-time, time-invariant dynamics [30]. As opposed to fixed abstraction approaches where the abstraction is computed prior to the discrete control protocol synthesis, which is performed only once, the sampling-based method progressively samples points in the continuous state space. Discrete control protocol synthesis then needs to be performed each time a sample point is added. This iterative procedure is terminated once a discrete plan satisfying a given specification is found. In order to make this procedure effective, the discrete control protocol synthesis is performed in an incremental manner based on the local model checking algorithm that appears in [36]. In this case, probabilistic completeness can be ensured, i.e., if there exists an under-approximation TS of the physical system for which φ is satisfiable, this method finds it with probability approaching one as the number of samples increases. Applications of the sampling-based approach to non-deterministic systems, however, have not been considered.

5. Model-Checking-Based Synthesis

For the case where the system is deterministic, the discrete control protocol synthesis problem is reduced to finding a path in TS that satisfies φ, which can be formulated as a non-emptiness of the specification or a satisfiability problem, also commonly known as a model checking problem [3]. Such a problem can be solved by claiming that $Trace(TS) \cap Words(\varphi) = \emptyset$. In case of negative result, a counterexample, which is a word in $Trace(TS) \cap Words(\varphi)$, can be used to obtain a path π of TS that satisfies φ. A positive result means $Trace(TS) \cap Words(\varphi) = \emptyset$, i.e., a path π of TS that satisfies φ does not exist; hence, we can conclude that φ is not satisfiable.

For simplicity of the presentation, we assume that TS only has one valid initial state. (For TS with multiple valid initial states, the procedure described below can be applied to each initial state separately.) To check whether $Trace(TS) \cap Words(\varphi) = \emptyset$, existing results from formal verification can be employed [3]. First, a non-deterministic Buchi automaton $\mathcal{A}_\varphi = (Q, \Sigma, \delta, Q_0, F)$ that accepts all and only words over AP that satisfy φ is computed. The product $TS_p = TS \otimes \mathcal{A}_\varphi$ can then be constructed based on the following definition.

Definition 5.1. Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system and $\mathcal{A}_\varphi = (Q, \Sigma, \delta, Q_0, F)$ be a non-deterministic Buchi automaton. The product of TS and $\mathcal{A}_\varphi$ is the transition system $TS_p = TS \otimes \mathcal{A}_\varphi$ defined by $TS_p = (S \times Q, Act, \rightarrow_p, I_p, Q, L_p)$ where

(i) for any $s, t \in S$, $\alpha \in Act$ and $p, q \in Q$, $\langle s, p\rangle \xrightarrow{\alpha}_p \langle t, q\rangle$ if and only if $s \xrightarrow{\alpha} t$ and $p \xrightarrow{L(t)} q$,
(ii) $I_p = \{\langle s_0, q_0\rangle : s_0 \in I$ and $\exists q \in Q_0$ such that $q \xrightarrow{L(s_0)} q_0\}$ and
(iii) $L_p : S \times Q \rightarrow 2^Q$ is given by $L_p(\langle s, q\rangle) = \{q\}$.

Consider a path $\pi_p = \langle s_0, q_0\rangle\langle s_1, q_1\rangle \ldots$ on TS_p. We say that π_p is *accepting* if and only if there exist infinitely many $j \geq 0$ such that $q_j \in F$. Stepping through Definition 5.1 shows that given a path π_p on TS_p, the corresponding path $\pi = s_0 s_1 \ldots$ on TS generates a word $L(s_0)L(s_1)\ldots$ that satisfies φ if and only if π_p is accepting. Hence, an accepting path of TS_p uniquely corresponds to a path of TS that satisfies φ. As a result, the model-checking-based synthesis can be reduced to a graph search problem to find a state $\langle s, q\rangle$ in TS_p satisfying the following conditions:

- $L_p(\langle s, q\rangle) \in F$.
- $\langle s, q\rangle$ is reachable, i.e., there exists a finite path fragment π_p^p from some $\langle s_0, q_0\rangle \in I_p$ to $\langle s, q\rangle$ in TS_p.
- $\langle s, q\rangle$ is on a direct cycle, i.e., there exists a finite path fragment π_p^c from some $\langle s', q'\rangle \in Post(\langle s, q\rangle)$ to $\langle s, q\rangle$ in TS_p.

If such $\langle s, q\rangle$ does not exists, we can conclude that φ is not satisfiable. Otherwise, an accepting path π_p on TS_p can be simply defined by $\pi_p = \pi_p^p(\pi_p^c)^\omega$. Let $\pi = s_0 s_1 \ldots$ be the path on TS corresponding to π_p. We define a control protocol $\mathcal{C}$ for TS by

$$\mathcal{C}(s'_0 s'_1 \ldots s'_i) = \begin{cases} \alpha_i \text{ if } s'_0 s'_1 \ldots s'_i = s_0 s_1 \ldots s_i, \\ \alpha'_i \text{ otherwise,} \end{cases}$$

where $s_i \xrightarrow{\alpha_i} s_{i+1}$ and $\alpha'_i \in Act(s'_i)$ can be picked arbitrarily. It can be easily checked that by applying $\mathcal{C}$, the system can satisfy φ by always picking the next state to follow path π.

Various model checkers have been developed for different specification languages. TLC [37] is a model checker for specifications written in TLA+, which is a specification language based on Temporal Logic of Actions (TLA) [38]. TLA introduces new kinds of temporal assertions to traditional linear temporal logic to make it practical to describe a system by a single formula and to make the specifications simpler and easier to understand. SPIN, on the other hand, is a model checker for specifications written in Process Meta-Language (PROMELA) [39]. Other popular model checkers include Symbolic Model Verifier (SMV) [40] and its successor NuSMV [41].

Computational complexity: As mentioned in Section 3.3, the number of states of $\mathcal{A}_\varphi$ is exponential in the length of φ. Hence, the number of states of the product transition system TS_p is $O(|S|)2^{O|\varphi|}$ where $|S|$ is the number of states in TS and $|\varphi|$ is the length of φ. A nested depth-first search algorithm [3] can be used to detect accepting cycles efficiently, with the worst-case time complexity that is linear in the number of states and transitions of TS_p.

Example 10. Consider the traffic light system $TS = T_1||T_2$ shown in Fig. 3 and the desired property $\varphi = \Box(\neg g_1 \vee \neg g_2) \wedge \Box\Diamond g_1 \wedge \Box\Diamond g_2$, i.e., we want to make sure that the two lights are never green at the same time and each light turns green infinitely often. A non-deterministic Buchi automaton $\mathcal{A}_\varphi$ that recognizes φ and the product transition system $TS_p = TS \otimes \mathcal{A}_\varphi$ are shown in Fig. 12 and Fig. 13, respectively. Comparing to Fig. 6, the projection of the accepting path shown in Fig. 13 onto the state of TS is exactly the same as the transition system induced by applying the control protocol described in Example 5.

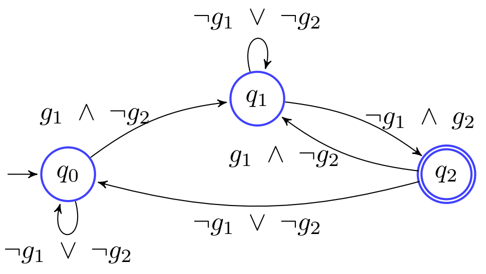


Fig. 12. Non-deterministic Buchi automata $\mathcal{A}_\varphi$ that recognize $\varphi = \Box(\neg g_1 \vee \neg g_2) \wedge \Box\Diamond g_1 \wedge \Box\Diamond g_2$. Accepted states are drawn with a double circle.

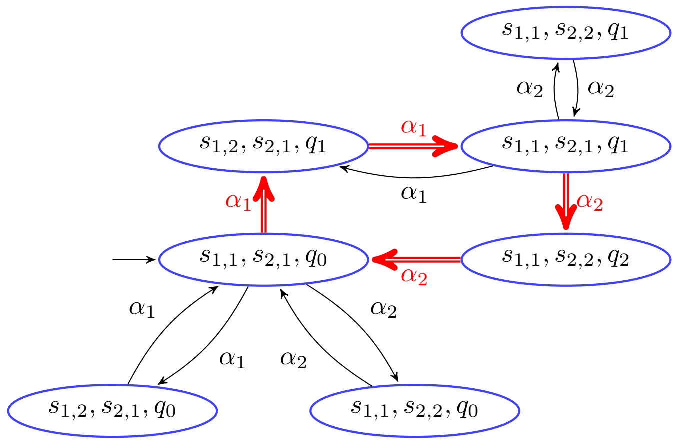


Fig. 13. The product transition system $TS_p = (T_1||T_2) \otimes \mathcal{A}_\varphi$, showing only reachable states. An accepting path is highlighted by double (red) arrows.

6. Reactive Module Synthesis

For the case where the system is non-deterministic, a control protocol needs to ensure that specification φ is satisfied for all the possible non-deterministic choices (e.g., all the possible behavior of the environment). As discussed in [6,7], the control protocol synthesis in this case can be treated as a two-player game between the system and the environment (i.e., adversary): the system and the environment alternatingly pick their actions: the environment attempts to falsify φ while the system attempts to satisfy φ. A provably correct control protocol therefore needs to ensure that φ is satisfied for any behavior of the environment and hence is represented by a satisfying tree where the branching represents all possible environment (i.e., non-deterministic) actions as shown in Fig. 14.

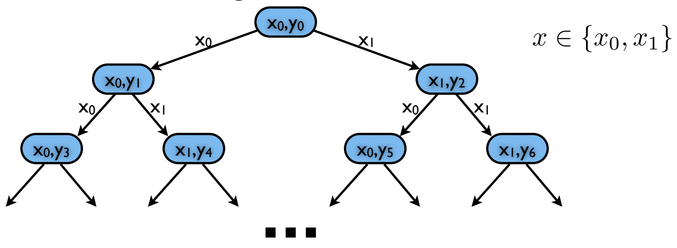


Fig. 14. A satisfying tree representing a control protocol for non-deterministic systems. The state of the system is a tuple (x, y) where $x \in \{x_0, x_1\}$ represents the non-deterministic choice of the environment behavior whereas $y \in \{y_0, y_1, \ldots\}$ represents the state over which the system has control.

Solving the above two-player game typically involves computing the winning set, which is defined as the set of initial states, starting from which there exists a strategy for the system to satisfy the specification for all the possible behavior of the environment. A procedure for computing the winning set also relies on computing the product of transition systems and finite automata as in the model-checking-based synthesis procedure, except that a non-deterministic Buchi automaton $\mathcal{A}_\varphi$ that recognizes φ needs to be "determinized" into a deterministic Rabin automaton $\mathcal{R}_\varphi$ [3]. The product $TS_p = TS \otimes \mathcal{R}_\varphi$, defined similar to Definition 5.1, is then computed. Finally, a fixed-point strategy can be applied on TS_p to isolate the winning set. A control protocol can then be constructed by saving intermediate values in the winning set computation. The running time of this synthesis algorithm as well as the size of TS_p are both at most double exponential in the length of φ. (The first exponent results from the construction of non-deterministic Buchi automaton $\mathcal{A}_\varphi$ from φ whereas the second exponent results from the determinization of $\mathcal{A}_\varphi$ into a deterministic Rabin automaton $\mathcal{R}_\varphi$.) We refer the reader to [6, 42] for more details.

It has been shown that for certain classes of specifications, such as those of the form $\Box p$, $\Diamond p$, $\Box\Diamond p$ and $\Diamond\Box p$ where p is a proposition, the synthesis problem can be solved with lower complexity [43, 44]. The main idea is to avoid the translation of the formula to a non-deterministic Buchi automaton and the determinization of the non-deterministic Buchi automaton into a deterministic Rabin automaton. For example, for the reachability game where the specification is of the form $\varphi = \Diamond p$, we define $W = \{s \in S : s \models p\}$ to be the set of states satisfying p and the predecessor operator $Pre_{\exists\forall} : 2^S \to 2^S$ where $Pre_{\exists\forall}(R)$ is the set of states whose all successors have at least one successor in R, i.e., $Pre_{\exists\forall}(R) = \{s \in S : \forall s' \in S \text{ if } s \to s', \text{ then } \exists s'' \in R \text{ such that } s' \to s''\}$. The set of all the states starting from which the controller can force the system into W can be computed efficiently by the iteration sequence

$$\begin{aligned} R_0 &= W, \\ R_i &= R_{i-1} \cup Pre_{\exists\forall}(R_{i-1}), \forall i > 0. \end{aligned}$$

From Tarski-Knaster Theorem, it can be shown that there exists a natural number n such that $R_n = R_{n-1}$. In addition, such R_n is the minimal solution of the fix-point equation $R = W \cup Pre_{\exists\forall}(R)$.

The methodology presented in [7], which is summarized below, allows us to solve a more general game efficiently through the use of binary decision diagrams. First, a *game structure* is defined as a tuple $\mathcal{G} = (V, X, Y, \theta_e, \theta_s, \rho_e, \rho_s, AP, L, \varphi)$ where

- V is a finite set of variables over finite domains,
- $X \subseteq V$ is a set of environment variables,
- $Y = V \setminus X$ is a set of controlled variables,
- $\theta_e(X)$ is a proposition over X characterizing the initial states of the environment,
- $\theta_s(V)$ is a proposition over V characterizing the initial states of the system,
- $\rho_e(V, X')$ is a proposition relating a state $s \in dom(V)$ to a possible next input value $s_X \in dom(X)$ and characterizes the transition relation of the environment,
- $\rho_s(V, X', Y')$ is a proposition relating a state $s \in dom(V)$ and an input value $s_X \in dom(X)$ to an output value $s_Y \in dom(Y)$ and characterizes the transition relation of the system,
- AP is a set of atomic propositions,
- $L : dom(V) \to 2^{AP}$ is a labeling function and
- φ is the winning condition, given by an LTL formula.

We let $dom(V)$, $dom(X)$ and $dom(Y)$ denote the set of all the possible assignments to variables in V, X and Y, respectively. An environment state $s_X \in dom(X)$ is a valid input in state $s \in dom(V)$ if $(s, s_X) \models \rho_e$ whereas a controlled state $s_Y \in dom(Y)$ is a valid output in state $s \in dom(V)$ reading input s_X if $(s, s_X, s_Y) \models \rho_s$.

Example 11. For an autonomous driving problem, a game structure $\mathcal{G} = (V, X, Y, \theta_e, \theta_s, \rho_e, \rho_s, AP, L, \varphi)$ may be defined such that

- the variables in X capture the position of obstacles, other cars, pedestrians, etc.,
- the variables in Y capture the maneuver state of the vehicle, e.g., drive or stop, or whether passing or reversing is allowed, etc.,
- θ_e describes the valid initial states of the environment, e.g., where obstacles can be,
- θ_s describes the valid initial states of the vehicle, e.g., the stop state,
- ρ_e describes how obstacles may move,
- ρ_s describes the valid transitions of the maneuver states of the vehicle and
- φ describes the winning condition, e.g., vehicle does not get stuck.

A game is played as follows. Initially, the environment chooses an assignment $s_X \in dom(X)$ such that $s_X \models \theta_e$ and the system chooses an assignment $s_Y \in dom(Y)$ such that $(s_X, s_Y) \models \theta_e \wedge \theta_s$. From a state s, the environment chooses an input $s_X \in dom(X)$ such that $(s, s_X) \models \rho_e$ and the system chooses an output $s_Y \in dom(Y)$ such that $(s, s_X, s_Y) \models \rho_s$. Formally, we define a *play* as a maximal sequence of states $\sigma = s_0 s_1 \ldots$ such that $s_0 \models \theta_e \wedge \theta_s$ and for each $j \geq 0$, $(s_j, s_{j+1}) \models \rho_e \wedge \rho_s$.

A *finite memory control protocol* for the system can be identified with a partial function $f : M \times dom(V) \times dom(X) \to M \times dom(Y)$, where M is some memory domain with a designated initial value $m_0 \in M$, such that for every $s \in dom(V)$, $s_X \in dom(X)$ and $m \in M$, if $(s, s_X) \models \rho_e$ and $f(m, s, s_X) = (m', s_Y)$, then $(s, s_X, s_Y) \models \rho_s$. Protocol f is winning for the system starting from state s_0 if any play $\sigma = s_0 s_1 \ldots$ with $f(m_i, s_i, s_{i+1}|_X) = (m_{i+1}, s_{i+1}|_Y), \forall i \geq 0$ either (i) is infinite and satisfies φ, or (ii) is finite and there is no assignment $s_X \in dom(X)$ such that $(s_n, s_X) \models \rho_e$ where s_n is the last state in σ. We let Win_s denote a proposition characterizing the set of states starting from which there exists a winning strategy for the system. A game structure is *winning for the system* if for each $s_X \in dom(X)$ such that $s_X \models \theta_e$, there exists $s_Y \in dom(Y)$ such that $(s_X, s_Y) \models \theta_s$ and $(s_X, s_Y) \in Win_s$.

For certain LTL specifications, μ-calculus over game structures can be employed to characterize the set of winning states of the system. The description of μ-calculus, however, is beyond the scope of this paper and we refer the reader to [7, 45]. As an example, μ-calculus formula $\mu R(p \vee \circleddiamond R)$ characterizes the set of states from which system can force the game to eventually visit p-states; hence, provides the solution for the reachability game previously discussed. Here, μ is the least fixpoint operator in μ-calculus, R is known as a "relational variable" and the operator $\circleddiamond$ is defined roughly similar to the predecessor operator $Pre_{\exists\forall}$.

Reference [7] considers a wider class of LTL formula known as *Generalized Reactivity[1]* (GR[1]), which covers LTL formulas of the form

$$\varphi = (\Box\Diamond p_1 \wedge \ldots \wedge \Box\Diamond p_m) \implies (\Box\Diamond q_1 \wedge \ldots \wedge \Box\Diamond q_n). \tag{10}$$

Roughly, the left hand side of $\implies$ specifies the assumption on the environment behavior whereas the right hand side of $\implies$ specifies the desired property of the system. [7] shows that there exists a μ-calculus formula that characterizes the set of winning states of the system for GR[1] winning condition, allowing the synthesis problem to be solved based on fixpoint computation in time proportional to $nm|dom(V)|^3$ where $|dom(V)|$ is the size of the state space. The proposed synthesis procedure has been implemented in JTLV [7]. We refer the reader to [7] for more details, including the discussion on the expressiveness of GR[1] and an extension to handle formulas of the form $\varphi_e \implies \varphi_s$ where φ_e and φ_s are any LTL formulas that can be represented by a deterministic Buchi automaton, which is defined as non-deterministic Buchi automaton (see Definition 3.5) with additional constraints that $|Q_0| \leq 1$ and for any $q \in Q$ and $\sigma \in \Sigma$, $(q, \sigma, q') \in \delta$ and $(q, \sigma, q'') \in \delta$ imply that $q' = q''$. LTL formulas that can be represented by a deterministic Buchi automaton include those of the form $\Box(p_1 \implies \Diamond p_2)$ where p_1 and p_2 are propositions.

7. Receding Horizon Temporal Logic Planning

The main limitation of the discrete synthesis described in Section 5 and Section 6 is the state explosion problem. In the worst case, all the possible states of the system have to be taken into account. For example, if the system has $|V|$ variables, each can take any of the P possible values, then the state space may contain as many as $P^{|V|}$ states.

To partially alleviate the state explosion problem, in [8, 11], we consider reactive module synthesis with GR[1] specifications and show that for systems with a certain structure, the synthesis problem can be solved in a receding horizon fashion, i.e., compute the plan or strategy over a "shorter" horizon, starting from the current state, implement the initial portion of the plan, move the horizon one step ahead and recompute. This approach essentially reduces the discrete control protocol synthesis problem into a set of smaller problems. The size of these smaller problems depends on the horizon length. For example, consider the autonomous driving problem where an autonomous vehicle needs to navigate the road shown in Fig. 15, starting from cell $C_{1,1}$ with the destination $C_{1,L} \cup C_{2,L} \cup C_{3,L}$. Suppose the horizon length is l (i.e., the vehicle plans for l cells ahead). Then, the state space for each short-horizon problem contains at most $3l2^{3l}$ states (whereas the size of the original problem is $3L2^{3L}$). Hence, the horizon length should be made as small as possible, subject to the realizability of the resulting short-horizon specifications as too short horizon typically renders the specifications unrealizable.

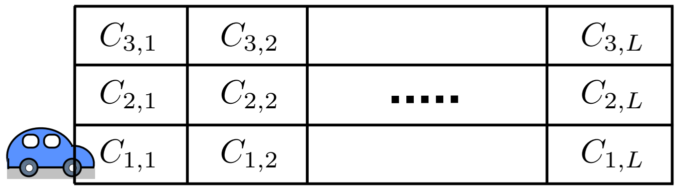


Fig. 15. The autonomous driving example where the road is partitioned into $3L$ cells where L is the length of the road.

Sufficient conditions that ensure that this receding horizon implementation preserves the desired system-level properties are presented in [8, 11]. For the simplicity of the presentation, in this article, we consider the case where the specification is given by

$$\varphi = (\varphi_{init} \wedge \varphi_{env}) \implies (\varphi_{safety} \wedge \varphi_{goal}), \tag{11}$$

where φ_{init} is a proposition characterizing the set of initial states, φ_{env} is an LTL formula characterizing the assumption on the environment behavior and can be written as the conjunction of a safety formula and the progress formulas on the left hand side of $\implies$ in (10), φ_{safety} is a safety formula and φ_{goal} is of the form $\varphi_{goal} = \Box\Diamond q$ where q is a proposition characterizing the set of "goal" states to be visited infinitely often.

The receding horizon approach works as follows. First, we organize the discrete state space into a partially ordered set $(\{\mathcal{W}_0, \ldots, \mathcal{W}_M\}, \prec_\varphi)$ such that $\mathcal{W}_0$ only contains the goal states and $\mathcal{W}_0 \prec_\varphi \mathcal{W}_i$ for all $i \neq 0$. A map $\mathcal{F} : \{\mathcal{W}_0, \ldots, \mathcal{W}_M\} \to \{\mathcal{W}_0, \ldots, \mathcal{W}_M\}$, which captures the horizon length, then needs to be defined such that $\mathcal{F}(\mathcal{W}_i) \prec_\varphi \mathcal{W}_i$ for all $i \neq 0$. Finally, we specify a proposition Φ that characterizes the receding horizon invariant such that any state that satisfies φ_{init} also satisfies Φ (i.e., $\varphi_{init} \implies \Phi$ is a tautology). We then define a short-horizon specification Ψ_i associated with each $\mathcal{W}_i, i \in \{0, \ldots, M\}$ as

$$\Psi_i = \big((\nu \in \mathcal{W}_i) \wedge \Phi \wedge \varphi_{env}\big) \implies \big(\Box\Phi \wedge \varphi_{safety} \wedge \Diamond(\nu \in \mathcal{F}(\mathcal{W}_i))\big) \tag{12}$$

where ν is the state of the system. This short-horizon specification essentially states that (a) the initial state is assumed to be in $\mathcal{W}_i$ and satisfies Φ, (b) the environment is assumed to satisfy the assumptions stated in the original specification and (c) the original safety properties are satisfied, Φ holds throughout an execution and the system eventually reaches a state in $\mathcal{F}(\mathcal{W}_i)$. Hence, $\mathcal{F}(\mathcal{W}_i)$ essentially defines an intermediate goal for states in $\mathcal{W}_i$. In addition, Φ

is introduced to ensure that a provably correct plan exists when the system reaches the end of the current horizon and needs to compute a new plan. We refer the reader to [8] for a detailed discussion on this receding horizon framework, including an extension to the case where there are multiple goals that may be visited in an arbitrary order.

Consider a simple example shown in Fig. 16 where ν_{10} is the goal state. The partial order may be defined as $\mathcal{W}_0 \prec_\varphi \mathcal{W}_1 \prec_\varphi \ldots \prec_\varphi \mathcal{W}_4$ and the map $\mathcal{F}$ may be defined as $\mathcal{F}(\mathcal{W}_j) = \mathcal{W}_{j-2}$, for all $j \geq 2$ and $\mathcal{F}(\mathcal{W}_1) = \mathcal{F}(\mathcal{W}_0) = \mathcal{W}_0$. The key idea of the receding horizon framework is to synthesize a control protocol for short-horizon specification Ψ_4, which corresponds to going from ν_1 only to a state in $\mathcal{F}(\mathcal{W}_4) = \mathcal{W}_2$, rather than synthesizing a control protocol for going from the initial state ν_1 to the goal state ν_{10} in one shot, taking into account all the possible behavior of the environment. Once a state in $\mathcal{W}_3$, i.e., ν_5 or ν_6 is reached, we then recompute a protocol for the short-horizon specification Ψ_3 for going to a state in $\mathcal{F}(\mathcal{W}_3) = \mathcal{W}_1$. This process is then continually repeated. From the finiteness of the set $\{\mathcal{W}_0, \ldots, \mathcal{W}_M\}$ and its partial order, it can be shown that this receding horizon implementation of the short-horizon strategies ensures the correctness of the global specification, provided that all the short horizon specifications $\Psi_i, i \in \{0, \ldots M\}$ are realizable [8]. In this case, the invariant Φ is introduced to rule out the states that render the short horizon problems unrealizable.

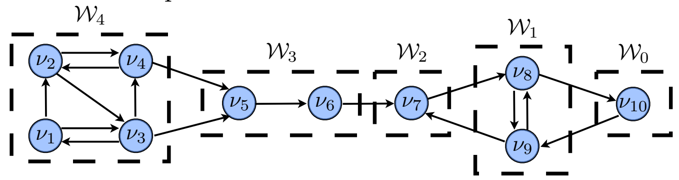


Fig. 16. A graphical description of the receding horizon framework for a special case where there is only one goal ν_{10}. $\nu_1, \ldots, \nu_{10}$ are the discrete states.

Given a receding horizon invariant Φ, the partial order $\prec_\varphi$ as well as the horizon length defined by the map $\mathcal{F}$ can be automated by adding an additional component, namely the "goal generator" to the hierarchical control structure in Fig. 8. The goal generator works on a graph $\mathbb{G}$ with $\mathcal{W}_i, i \in \{0, \ldots M\}$ being its states. For each $i, j \in \{0, \ldots M\}$, a transition from $\mathcal{W}_i$ to $\mathcal{W}_j$ in $\mathbb{G}$ is added if $i \neq j$ and the short horizon specification Ψ_i is realizable with $\mathcal{F}(\mathcal{W}_i) = \mathcal{W}_j$. After $\mathbb{G}$ is constructed, the goal generator then performs a graph search to find a path from $\mathcal{W}_i$ to which the current state of the system belongs to a goal state in $\mathcal{W}_0$. This path essentially defines a sequence of intermediate goals, each for each short horizon problem. The resulting hierarchical control structure with this implementation of the receding horizon framework is shown in Fig. 17. Fig. 18 shows the similarity of this hierarchical control structure with that implemented on Alice, illustrating that the techniques presented in this article can be utilized to formalize and enable automatic design of the navigation protocol stack of an autonomous system.

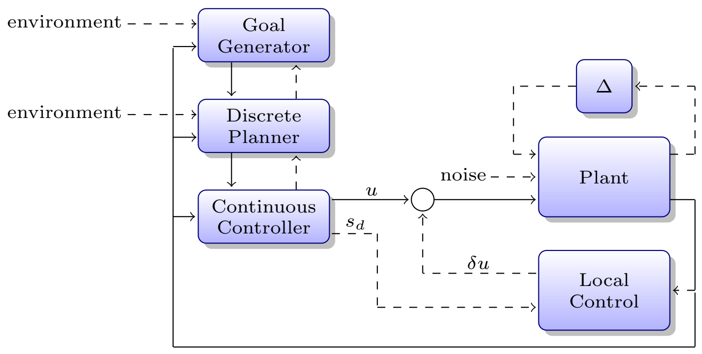
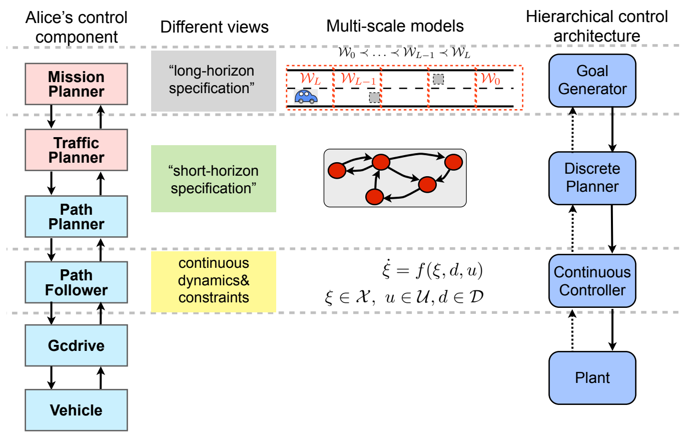


Fig. 17. The hierarchical control structure with the goal generator.



Alice's control component
Different views
Multi-scale models
Hierarchical control architecture
Mission Planner
"long-horizon specification"
$\mathcal{W}_0 \prec \ldots \prec \mathcal{W}_{L-1} \prec \mathcal{W}_L$
$\mathcal{W}_L$
$\mathcal{W}_{L-1}$
$\mathcal{W}_0$
Goal Generator
Traffic Planner
Path Planner
"short-horizon specification"
Discrete Planner
Path Follower
continuous dynamics& constraints
$\dot{\xi} = f(\xi, d, u)$
$\xi \in \mathcal{X},\ u \in \mathcal{U}, d \in \mathcal{D}$
Continuous Controller
Gcdrive
Plant
Vehicle

Fig. 18. The hierarchical control structure with the receding horizon implementation, showing the similarity with the navigation protocol stack implemented on Alice. The goal generator has similar functionality as Mission Planner. It determines a sequence of intermediate goals for the discrete planner such that the original "long-horizon" specification is satisfied. Its computation relies on the graph $\mathbb{G}$ that encodes the partially ordered set $(\{\mathcal{W}_0, \ldots, \mathcal{W}_M\}, \prec_\varphi)$. The discrete planner has similar functionality as the composition of Traffic Planner and Path Planner. It computes a discrete plan for the system such that the short-horizon specification in (12) with the next intermediate goal computed by the goal generator is satisfied based on a finite, abstract model of the physical system. Finally, the continuous controller deals with the continuous dynamics and constraints to ensure that the physical system follows the plan computed by the discrete planner. This functionality is similar to that of Path Follower in Alice.

Computational Complexity and Completeness: The receding horizon implementation reduces the computational complexity by restricting the state space considered in each subproblem. However, it is not complete. Even if the original specification is realizable, there may not exist a combination of horizon length, partial order relation and receding horizon invariant that render all the short horizon specification realizable. Nevertheless, its successful applications to autonomous driving problems have been illustrated in [8–11]. Examples of these applications are provided in Figure 20.

Remark 7.1. Computation of the horizon length, partial order relation and receding horizon invariant requires insights for each problem domain. Automatic construction of these elements is subject to current research. Reference [8] describes automatic construction of certain elements, given other elements, e.g., automatic computation of the horizon length and partial order relation, given a receding horizon invariant, and automatic computation of the receding horizon invariant, given a horizon length and partial order relation.

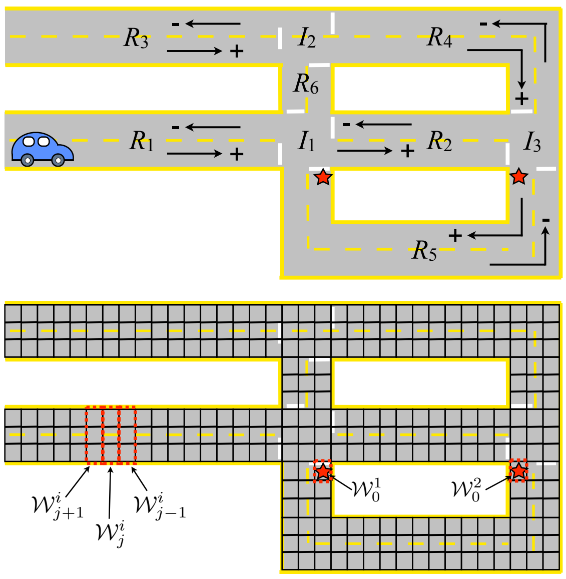


Fig. 19. The road network and its partition for the autonomous vehicle example. The stars indicate the cells that need to be visited infinitely often.

Example 12. Consider an autonomous driving problem in an urban-like environment. We consider the road network shown in Fig. 19, which is partitioned into $N = 282$ cells. Each of these cells may or may not be occupied by an obstacle. The desired properties include:

- Each of the two cells marked by star needs to be visited infinitely often.
- No collision is allowed, i.e., the vehicle cannot occupy the same cell as an obstacle
- The vehicle stays in the right lane unless there is an obstacle blocking the lane.
- The vehicle can only proceed through an intersection when the intersection is clear.

In [8], we show that with some mild assumptions on the environment behavior, there exists a receding horizon invariant that ensures that all the short-horizon specifications are realizable with horizon length 2, i.e, $\mathcal{F}(\mathcal{W}_i) = \mathcal{F}(\mathcal{W}_{i-2})$. Hence, the size of the state space for each short-horizon problem is at most 4608 whereas the size of the state space of the original problem is in the order of 10^{87}. Roughly, the receding horizon invariant requires that the vehicle is not surrounded by obstacles and if the vehicle is not in the travel lane, there must be an obstacle blocking the lane. Using JTLV, each short horizon synthesis problem can be solved in approximately 1.5 seconds on a MacBook with a 2 GHz Intel Core 2 Duo processor and 4 Gb of memory. Simulation results when the receding horizon approach is applied are shown in Fig. 20.

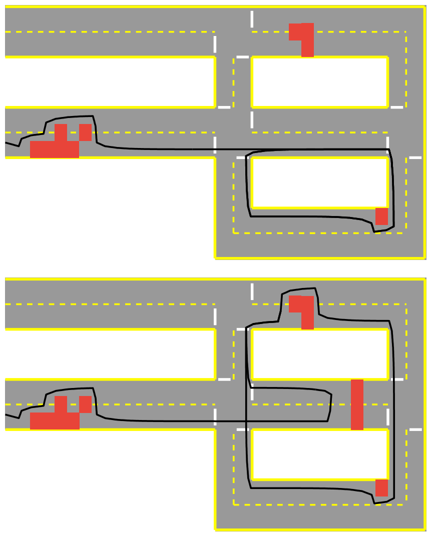


Fig. 20. Simulation results with (top) no road blockage, (bottom) a road blockage on the middle road. The corresponding movies can be downloaded from `http://sourceforge.net/projects/tulip-control/`.

8. TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning

TuLiP [12] is a Python-based toolbox for control protocol synthesis with LTL specifications. It takes, as an input, the model and the specification of the system. Currently, TuLiP only handles GR[1] specifications and systems modeled either by a finite transition system or by a discrete-time piecewise affine dynamics.

The key features of TuLiP include control protocol synthesis and receding horizon planning. The synthesis feature, as summarized in Fig. 21, relies on generating a proposition preserving partition of the continuous state space, partition refinement based on finite-time reachability analysis and reactive module synthesis. JTLV [7] is used as the underlying discrete control protocol synthesis routine.

In the receding horizon planning implementation, TuLiP automatically constructs the short horizon specification Ψ_i in (12) for each $i \in \{0, \dots, M\}$, given a partition of the discrete state space $\{\mathcal{W}_0, \dots, \mathcal{W}_M\}$, a map $\mathcal{F}$ and a receding horizon invariant Φ. It also includes functions for verifying that there exists a partial order $\prec_\varphi$ and that the sufficient condition for the correctness of the receding horizon implementation is satisfied, and computing the receding horizon invariant Φ if one exists or reporting an error otherwise.

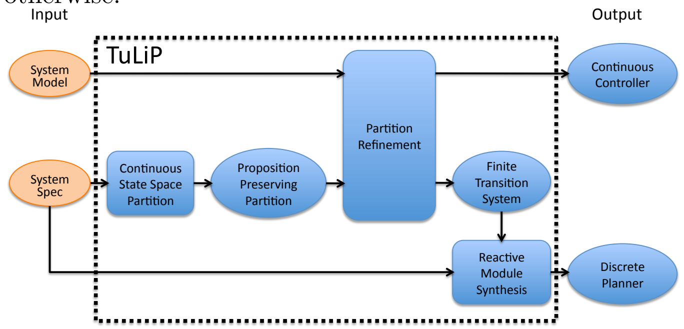


Fig. 21. The control protocol synthesis procedure implemented in TuLiP.

Successful applications of TuLiP include autonomous driving [9,11], vehicle management systems in avionics [46], multi-target tracking [47] and robotic manipulation [48]. Other simpler examples are included in the current release of the toolbox.

9. Concluding Remarks and Open Problems

In this article, we summarized existing results from formal methods and hybrid systems that enable automatic synthesis of control protocols with correctness guarantee for finite transition systems, dynamical systems and hybrid systems. Non-determinism was used to capture uncertainties in the system, especially those arising from different choices of valid environment behavior over which the system does not have any control. Correctness of the system was characterized by temporal logic formulas that express the desired properties of the system as well as the assumptions on the unknowns or environment.

While several application domains have demonstrated the promising potential of these control protocol synthesis techniques, a number of problems remain open. We here focus on scalability, optimality and robustness.

Computation complexity: Even though the discrete synthesis described in this article can be performed in polynomial time for a certain class of properties, for practical problems the rapid increase in computational complexity with the size of the state space is one of the limiting factors that restricts the application of correct-by-construction synthesis. The receding horizon framework addresses this challenge by decomposing the global reactive synthesis problem into smaller, "short-horizon" synthesis problems. The decomposition is driven by a partial order structure, induced by the mission specifications, embedded in the discrete state space. Another possibility for decomposition is the underlying networked system structure. In that case, it may be possible to compositionally design distributed controllers for each part of the system in such a way that the controllers, when implemented together, work correctly for the overall system [47,49]. Synthesis can then be performed separately for each individual subsystem provided that appropriate information exchange and interface models can be found such that each local specification is realizable. Closely related to [47, 49] is the work reported in [50–52].

Another recent approach, namely "incremental synthesis," enables the synthesis to be performed in an anytime-fashion, taking into account only a small number of subsystems initially and successively adding more subsystems to the synthesis procedure until the computational resource constraints are exceeded [53, 54]. This approach allows a suboptimal solution to be obtained in the case where the full problem cannot be solved due to the state explosion problem.

Optimality: In many applications, not only the correctness but the optimality of the system is also critical. The synthesis of optimal correct control protocols have been considered for deterministic and probabilistic systems [55–58]. [59] considers mean-payoff parity games on graphs. Such games incorporate both quantitative and qualitative objectives where the qualitative component is a parity condition and the quantitative component is a mean-payoff reward. Dealing with optimality in non-deterministic systems with more general classes of quantitative and qualitative objectives, however, remains an open problem.

Accounting for uncertainties: In this article, uncertainties are captured as non-determinism in the system and reactive module synthesis is employed to generate a correct control protocol. On the other hand, these methods do not provide any quantitative information of the robustness of the synthesized system against uncertainties. For example, in the case where the specification is not realizable, one may want to know the probability of failure. To provide such quantitative information, probabilistic systems are considered in [60] where the system is modeled as a Markov decision process and the objective of the synthesis is to generate a control protocol that maximizes the probability that the system satisfies a given specification. However, open issues remain, for example, regarding how to generate a good probabilistic model of the system. An initial attempt toward handling modeling uncertainties is in [61] where strategies for Markov decision processes with uncertain transition probabilities are synthesized from temporal logic specifications.

An important yet implicit assumption in reactive synthesis is that the sensing and perception information is complete and perfect. Some initial attempts to deal with partially observable systems is in [62–64].

References

[1] M. Buehler, K. Iagnemma and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic* (Springer Verlag, 2010).

[2] J. Markoff, Google cars drive themselves, in traffic The New York Times, (October 9, 2010).

[3] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)* (The MIT Press, 2008).

[4] E. A. Emerson, Temporal and modal logic, *Handbook of Theoretical Computer Science (Vol. B): Formal Models and Semantics* (1990) 995–1072.

[5] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems* (Springer-Verlag, 1992).

[6] A. Pnueli and R. Rosner, On the synthesis of a reactive module, *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (ACM, 1989), pp. 179–190.

[7] N. Piterman, A. Pnueli and Y. Sa'ar, *Synthesis of Reactive(1) Designs, Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science, Vol. 3855 (Springer Berlin Heidelberg, 2005), ch. 24, pp. 364–380. Software available at `http://jtlv.sourceforge.net/`.

[8] T. Wongpiromsarn, U. Topcu and R. M. Murray, Receding horizon temporal logic planning, *IEEE Transactions on Automatic Control* **57**(11) (2012) 2817–2830.

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

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

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

[12] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu and R. M. Murray, TuLiP: A software toolbox for receding horizon temporal logic planning, *International Conference on Hybrid Systems: Computation and Control*, (2011). software available at `http://www.cds.caltech.edu/tulip`.

[13] J. W. Burdick, N. DuToit, A. Howard, C. Looman, J. Ma, R. M. Murray and T. Wongpiromsarn, Sensing, navigation and reasoning technologies for the DARPA Urban Challenge, tech. rep., DARPA Urban Challenge Final Report (2007).

[14] N. E. DuToit, T. Wongpiromsarn, J. W. Burdick and R. M. Murray, Situational reasoning for road driving in an urban environment, *International Workshop on Intelligent Vehicle Control Systems*, (2008).

[15] T. Wongpiromsarn and R. M. Murray, Distributed mission and contingency management for the DARPA Urban Challenge, *International Workshop on Intelligent Vehicle Control Systems*, (2008).

[16] T. Wongpiromsarn, S. Mitra, R. M. Murray and A. Lamperski, Verification of periodically controlled hybrid systems: Application to an autonomous vehicle, *ACM Transactions in Embedded Computing Systems, Special Issue on the Verification of Cyber-Physical Software Systems* **11** (2012).

[17] A. Pnueli, The temporal logic of programs, *Proc. of the 18th Annual Symposium on the Foundations of Computer Science*, (IEEE, 1977), pp. 46–57.

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

[19] A. Galton (ed.), *Temporal Logics and Their Applications* (Academic Press Professional, Inc., San Diego, CA, 1987).

[20] F. Lin, Analysis and synthesis of discrete event systems using temporal logic, *Control Theory and Advanced Technologies* **9**(1) (1993) 341–350.

[21] S. Jiang and R. Kumar, Failure diagnosis of discrete event systems with linear-time temporal logic fault specifications, *IEEE Transactions on Automatic Control* **49**(6) (2004) 934–945.

[22] R. Koymans, Specifying real-time properties with metric temporal logic, *Real-Time Syst.* **2**(4) (1990) 255–299.

[23] P. Gastin and D. Oddoux, Fast LTL to Büchi automata translation, *Proceedings of the 13th International Conference on Computer Aided Verification*, eds. G. Berry, H. Comon and A. Finkel *Lecture Notes in Computer Science* **2102**, (Springer, Paris, France, 2001), pp. 53–65. Software available at `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`.

[24] R. Alur, T. A. Henzinger, G. Lafferriere and G. J. Pappas, Discrete abstractions of hybrid systems, *Proceedings of the IEEE* **88**(7) (2000) 971–984.

[25] P. Tabuada and G. J. Pappas, Linear time logic control of discrete-time linear systems, *IEEE Transactions on Automatic Control* **51**(12) (2006) 1862–1877.

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

[27] M. Kloetzer and C. Belta, A fully automated framework for control of linear systems from temporal logic specifications, *IEEE Transactions on Automatic Control* **53**(1) (2008) 287–297.

[28] A. Girard and G. J. Pappas, Hierarchical control system design using approximate simulation, *Automatica* **45**(2) (2009) 566–571.

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

[30] S. Karaman and E. Frazzoli, Sampling-based motion

planning with deterministic μ-calculus specifications, *Proc. of IEEE Conference on Decision and Control*, (2009), pp. 2222–2229.

[31] H. Tanner and G. J. Pappas, Simulation relations for discrete-time linear systems, *Proc. of the IFAC World Congress on Automatic Control*, (2002), pp. 1302–1307.

[32] D. Peled and T. Wilke, Stutter-invariant temporal properties are expressible without the next-time operator, *Information Processing Letters* **63**(5) (1997) 243–246.

[33] A. Girard, A. A. Julius and G. J. Pappas, Approximate simulation relations for hybrid systems, *Discrete Event Dynamic Systems* **18**(2) (2008) 163–179.

[34] J. Liu, N. O. Ufuk Topcud and R. M. Murray, Synthesis of reactive control protocols for differentially flat systems, *Proc. of IEEE Conference on Decision and Control*, (2012).

[35] T. Kalmr-Nagy, R. D'Andrea and P. Ganguly, Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle, *Robotics and Autonomous Systems* **46**(1) (2004) 47 – 64.

[36] K. Schneider, *Verification of Reactive Systems* (Springer, 2004).

[37] Y. Yu, P. Manolios and L. Lamport, Model checking TLA+ specifications, *Conference on Correct Hardware Design and Verification Methods*, (1999), pp. 54–66.

[38] L. Lamport, The temporal logic of actions, *ACM Transactions on Programming Languages and Systems* **16** (May 1994) 872–923.

[39] G. J. Holzmann, *The Spin Model Checker* (Addison-Wesley, 2004).

[40] K. L. McMillan, *Symbolic Model Checking* (Kluwer Academic Publishers, 1993).

[41] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, Nusmv 2: An open-source tool for symbolic model checking, *CAV*, eds. E. Brinksma and K. G. Larsen *Lecture Notes in Computer Science* **2404**, (Springer, 2002), pp. 359–364.

[42] M. Kloetzer and C. Belta, Dealing with nondeterminism in symbolic control, *Proceedings of the 11th international workshop on Hybrid Systems: Computation and Control*, (Springer-Verlag, Berlin, Heidelberg, 2008), pp. 287–300.

[43] E. Asarin, O. Maler, A. Pnueli and J. Sifakis, Controller synthesis for timed automata, *IFAC Symposium on System Structure and Control*, (Elsevier, 1998), pp. 469–474.

[44] R. Alur and S. La Torre, Deterministic generators and games for LTL fragments, *ACM Transactions on Computational Logic* **5**(1) (2004) 1–25.

[45] D. Kozen, Results on the propositional μ-calculus, *Theoretical Computer Science* **27**(3) (1983) p. 333354.

[46] T. Wongpiromsarn, U. Topcu and R. M. Murray, Formal synthesis of embedded control software: Application to vehicle management systems, *AIAA Infotech@Aerospace*, (2011).

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

[48] S. Chinchali, S. C. Livingston, U. Topcu, J. W. Burdick and R. M. Murray, Towards formal synthesis of reactive controllers for dexterous robotic manipulation, *IEEE International Conference on Robotics and Automation (ICRA)*, (2012), pp. 5183–5189.

[49] N. Ozay, U. Topcu and R. M. Murray, Distributed power allocation for vehicle management systems,, *Proc. of IEEE Conference on Decision and Control*, (2012).

[50] D. Peled and S. Schewe, Practical distributed control synthesis, *INFINITY*, eds. F. Yu and C. Wang *EPTCS* **73** (2011), pp. 2–17.

[51] G. Katz, D. Peled and S. Schewe, Synthesis of distributed control through knowledge accumulation, *CAV*, eds. G. Gopalakrishnan and S. Qadeer *Lecture Notes in Computer Science* **6806**, (Springer, 2011), pp. 510–525.

[52] B. Finkbeiner and S. Schewe, Coordination logic, *CSL*, eds. A. Dawar and H. Veith *Lecture Notes in Computer Science* **6247**, (Springer, 2010), pp. 305–319.

[53] T. Wongpiromsarn, A. Ulusoy, C. Belta, E. Frazzoli and D. Rus, Incremental temporal logic synthesis of control policies for robots interacting with dynamic agents, *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, (2012).

[54] A. Ulusoy, T. Wongpiromsarn and C. Belta, Incremental control synthesis in probabilistic environments with temporal logic constraints, *Proc. of IEEE Conference on Decision and Control*, (2012).

[55] A. Ulusoy, S. L. Smith, X. C. Ding and C. Belta, Robust multi-robot optimal path planning with temporal logic constraints, *IEEE International Conference on Robotics and Automation (ICRA)*, (2012), pp. 4693–4698.

[56] S. L. Smith, J. Tumova, C. Belta and D. Rus, Optimal path planning for surveillance with temporal-logic constraints, *International Journal of Robotics Research* **30**(14) (2011) 1695–1708.

[57] X. C. Ding, S. L. Smith, C. Belta and D. Rus, Mdp optimal control under temporal logic constraints, *Proc. of IEEE Conference on Decision and Control*, (2011), pp. 532–538.

[58] E. M. Wolff, U. Topcu and R. M. Murray, Optimal control with weighted average costs and temporal logic specifications, *Proc. of Robotics: Science and Systems*, (2012).

[59] K. Chatterjee, T. Henzinger and M. Jurdzinski, Mean-payoff parity games, *LICS 05*, (June 2005).

[60] X. C. Ding, S. L. Smith, C. Belta and D. Rus, LTL control in uncertain environments with probabilistic satisfaction guarantees, *IFAC World Congress*, (2011).

[61] E. M. Wolff, U. Topcu and R. M. Murray, Robust control of uncertain markov decision processes with tem-

poral logic specifications, *Proc. of IEEE Conference on Decision and Control*, (2012).

[62] K. Chatterjee, L. Doyen, T. A. Henzinger and J.-F. Raskin, Algorithms for omega-regular games with imperfect information, *Proceedings of the 20th international conference on Computer Science Logic*, (Springer-Verlag, Berlin, Heidelberg, 2006), pp. 287–302.

[63] T. Wongpiromsarn and E. Frazzoli, Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications, *Proc. of IEEE Conference on Decision and Control*, (2012).

[64] R. Dimitrova and B. Finkbeiner, Counterexample-guided synthesis of observation predicates, *FORMATS*, eds. M. Jurdzinski and D. Nickovic *Lecture Notes in Computer Science* **7595**, (Springer, 2012), pp. 107–122.