Submitted, 2013 International Conference on Robotics and Automation (ICRA)
http://www.cds.caltech.edu/~murray/papers/wtm13-acc.html

Efficient reactive controller synthesis for a fragment of linear temporal logic

Eric M. Wolff, Ufuk Topcu, and Richard M. Murray

***Abstract*—Motivated by robotic motion planning, we develop a framework for control policy synthesis for both non-deterministic transition systems and Markov decision processes that are subject to temporal logic task specifications. We introduce a fragment of linear temporal logic that can be used to specify common motion planning tasks such as safe navigation, response to the environment, surveillance, and persistent coverage. This fragment is computationally efficient; the complexity of control policy synthesis is a doubly-exponential improvement over standard linear temporal logic for both non-deterministic transition systems and Markov decision processes. This improvement is possible since we compute directly on the original system, as opposed to the automata-based approach commonly used for linear temporal logic. We give simulation results for representative motion planning tasks and compare to generalized reactivity(1).**

I. Introduction

As autonomous vehicles and robots are used more widely, it is important for users to be able to accurately and concisely specify tasks. Additionally, given a task and a system, one would like to automatically synthesize a control policy that guarantees that the system will complete the specified task. In this context, we consider the problem of control policy synthesis in the presence of an adversarial environment that behaves either non-deterministically or probabilistically.

A widely used task specification language is linear temporal logic (LTL). LTL allows one to reason about how system properties change over time, and thus specify a wide variety of tasks, such as safety (always avoid B), guarantee (eventually visit A), persistence (eventually always stay in A), and recurrence (infinitely often visit A). While LTL is a powerful language for specifying system properties, the complexity of synthesizing a control policy that satisfies an LTL formula is doubly-exponential in the formula length for both non-deterministic and probabilistic systems [7], [23].

Temporal logics have been used to specify desired behaviors for robots and hybrid systems for which controllers can then be automatically synthesized. A common approach is to abstract the original continuous system as a finite discrete system, such as a non-deterministic transition system or Markov decision process (MDP). Sampling-based motion planning techniques can be used for nonlinear systems to create an approximate deterministic transition system, for which a satisfying control policy can be computed [15], [22]. A framework for abstracting a linear system as a discrete transition system and then constructing a control policy that guarantees that the original system satisfies an LTL specification is presented in [17]. Reactive control policies are synthesized for linear systems in the presence of a non-deterministic environment in [18]. A receding horizon framework is used in [25] to handle the blow-up in system size of the previous approach. Finally, control policies are created for a Markov decision process that represents a robot with noisy actuators in [8].

Motivated by robot motion planning, we introduce a fragment of LTL that can be used to specify tasks such as safe navigation, immediate response to the environment, surveillance, and persistent coverage. For this fragment, we create control policies in time polynomial in the size of the system by computing reachable sets directly on the original system (as opposed to on the product of the system and a property automaton). The underlying algorithms are quite simple and the approach scales well. Preliminary experiments indicate that it outperforms standard implementations of generalized reactivity(1) [21] on some motion planning problems.

There has been much interest in determining fragments of LTL that are computationally efficient to reason about. Fragments of LTL that have exponential complexity for control policy synthesis were analyzed in [1]. In the context of timed automata, certain fragments of LTL have been used for efficient control policy synthesis [20]. The generalized reactivity(1) fragment can express many tasks and control policies can be synthesized in polynomial time in the size of the system [5], [21]. This fragment is extended to generalized Rabin(1), which is the largest fragment of specifications for which control policy synthesis can be done efficiently [9].

The main contribution of this paper is the use of an expressive fragment of LTL for efficient control policy synthesis for non-deterministic transition systems and Markov decision processes. A unified approach for control policy synthesis is presented that covers representative tasks and modeling frameworks. The algorithms used are simple and do not require detailed understanding of automata theory or formal methods. The fragment that we use is effectively a Rabin acceptance condition, which allows us to compute directly on the system.

We introduce a system model, a specification language, and a problem statement in Section II. Background on acceptance conditions, reachable sets, and graph notation is in Section III. We create control policies for deterministic transition systems in Section IV, for non-deterministic transition systems in Section V, and MDPs in Section VI. Note that we will

Eric M. Wolff and Richard M. Murray are with the Department of Control and Dynamical Systems, California Institute of Technology, Pasadena, CA. Ufuk Topcu is with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA. The corresponding author is ewolff@caltech.edu.

delay the introduction of MDPs until Section VI. We compare the complexity of our models with LTL and generalized reactivity(1) in Section VII and give examples in Section IX. We conclude with directions for future work in Section X.

II. Problem Formulation

In this section we give background and introduce the main problem. An *atomic proposition* is a statement that is either $True$ or $False$. The cardinality of a set X is denoted by $|X|$.

A. System model

We use finite transition systems and MDPs (introduced in Section VI) to model the system behavior. In robotics, however, one is usually concerned with continuous systems. This gap is partially bridged by constructive procedures for exactly abstracting relevant classes of continuous systems as finite transition systems [3], [14]. Additionally, sampling-based methods, such as rapidly-exploring random trees [19] and probabilistic roadmaps [16], build a finite transition system that approximates a continuous system [15], [22].

Definition 1. A (finite) *non-deterministic transition system* (NTS) is a tuple $\mathcal{T} = (S, A, R, s_0, AP, L)$ consisting of a finite set of states S, a finite set of actions A, a transition function $R : S \times A \to 2^S$, an initial state $s_0 \in S$, a set of atomic propositions AP, and a labeling function $L : S \to 2^{AP}$.

Let $A(s)$ denote the set of available actions at state s. Denote the parents of the states in the set $S' \subseteq S$ by $Parents(S') := \{s \in S \mid \exists a \in A(s) \text{ and } R(s,a) \cap S' \neq \varnothing\}$. The set $Parents(S')$ includes all states in S that can (possibly) reach S' in a single transition.

We assume that the transition system is non-blocking, i.e., $|R(s,a)| \geq 1$ for each state $s \in S$ and action $a \in A(s)$.

A *deterministic transition system* (DTS) is a non-deterministic transition system where $|R(s,a)| = 1$ for each state $s \in S$ and action $a \in A(s)$.

A *run* $\sigma = s_0 s_1 s_2 \dots$ of the transition system is an infinite sequence of its states, where $s_i \in S$ is the state of the system at index i (also denoted σ_i) and for each $i = 0, 1, \dots$, there exists $a \in A(s_i)$ such that $s_{i+1} \in R(s_i, a)$. A *word* is an infinite sequence of labels $L(\sigma) = L(s_0)L(s_1)L(s_2)\dots$ where $\sigma = s_0 s_1 s_2 \dots$ is a run.

A *memoryless control policy* for a non-deterministic transition system $\mathcal{T}$ is a map $\mu : S \to A$, where $\mu(s) \in A(s)$ for state $s \in S$. A *finite-memory control policy* is a map $\mu : S \times M \to A \times M$ where the finite set M is called the memory and $\mu(s,m) \in A(s) \times M$ for state $s \in S$ and mode $m \in M$. A control policy selects an action deterministically.

Given a state $s \in S$ and action $a \in A(s)$, there may be multiple possible successor states in the set $R(s,a)$, i.e., $|R(s,a)| > 1$. A single successor state $t \in R(s,a)$ is non-deterministically selected. We interpret this selection as an uncontrolled (adversarial) environment resolving the non-determinism.

The set of runs of $\mathcal{T}$ with initial state $s \in S$ induced by a control policy μ is denoted by $\mathcal{T}^{\mu}(s)$.

B. Linear temporal logic

We use a fragment of linear temporal logic (LTL) to concisely and unambiguously specify desired system behavior. We begin by defining LTL, from which our fragment will inherit syntax and semantics. A comprehensive treatment of LTL is given in [2].

Syntax: LTL includes: (a) a set of atomic propositions, (b) the propositional connectives: $\neg$ (negation) and $\wedge$ (conjunction), and (c) the temporal modal operators: $\bigcirc$ (next) and $\mathcal{U}$ (until). Other propositional connectives such as $\vee$ (disjunction) and $\implies$ (implication) and other temporal operators such as $\Diamond$ (eventually), $\Box$ (always), $\Box\Diamond$ (infinitely often), and $\Diamond\Box$ (eventually forever) can be derived.

An LTL formula is defined inductively as follows: (1) any atomic proposition is an LTL formula, (2) given formulas φ_1 and φ_2, $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\bigcirc\varphi_1$, and $\varphi_1 \,\mathcal{U}\, \varphi_2$ are LTL formulas.

Semantics: An LTL formula is interpreted over an infinite sequence of states. Given an infinite sequence of states $\sigma = s_0 s_1 s_2 \dots$ and a formula φ, the semantics are defined inductively as follows: (i) for atomic proposition p, $s_i \vDash p$ if and only if (iff) $p \in L(s_i)$; (ii) $s_i \vDash \neg\varphi$ iff $s_i \nvDash \varphi$; (iii) $s_i \vDash \varphi \wedge \psi$ iff $s_i \vDash \varphi$ and $s_i \vDash \psi$; (iv) $s_i \vDash \bigcirc\varphi$ iff $s_{i+1} \vDash \varphi$; and (v) $s_i \vDash \varphi \,\mathcal{U}\, \psi$ iff $\exists j \geq i$ s.t. $s_j \vDash \psi$ $\forall k \in [i, j), s_k \vDash \varphi$.

A *propositional formula* ψ is composed of only atomic propositions and propositional connectives. We denote the set of states where ψ holds by $[\psi]$.

An infinite sequence of states $\sigma = s_0 s_1 s_2 \dots$ *satisfies* the LTL formula φ, denoted by $\sigma \vDash \varphi$, if $s_0 \vDash \varphi$. The system $\mathcal{T}$ under control policy μ *satisfies* the LTL formula φ at state $s \in S$, denoted $\mathcal{T}^{\mu}(s) \vDash \varphi$ if and only if $\sigma \vDash \varphi$ for all $\sigma \in \mathcal{T}^{\mu}(s)$. Given a system $\mathcal{T}$, state $s \in S$ is *winning* for φ if there exists a control policy μ such that $\mathcal{T}^{\mu}(s) \vDash \varphi$. Let $W \subseteq S$ denote the set of winning states.

C. Problem Statement

We now formally state the main problem of the paper and give an overview of our solution approach.

Problem 1. Given a non-deterministic transition system $\mathcal{T}$ with initial state s_0 and an LTL formula φ, determine whether there exists a control policy μ such that $T^{\mu}(s_0) \vDash \varphi$. Return the control policy μ if it exists.

Problem 1 is intractable in general. Determining if there exists such a control policy takes time doubly-exponential in the length of φ [23]. Thus, we consider a fragment of LTL for which polynomial time solutions to Problem 1 exist. We will introduce such a fragment in Section II-D and solve Problem 1 for formulas of this form. We begin by solving Problem 1 for the special case of a deterministic transition system in Section IV. While this discussion is subsumed by that for the non-deterministic transition system, the lack of non-determinism allows for stronger results. We solve Problem 1 for non-deterministic transition systems in Section V. Finally, we solve an analogous problem for MDPs in Section VI.

D. A fragment of LTL

We now introduce a fragment of LTL that can specify a wide range motion planning tasks such as safe navigation, immediate response to the environment, surveillance, and persistent coverage.

We consider formulas of the form

$$\varphi = \varphi_{safe} \wedge \varphi_{act} \wedge \varphi_{per} \wedge \varphi_{rec}, \tag{1}$$

where

$$\varphi_{safe} := \Box p_1, \qquad \varphi_{act} := \bigwedge_{j \in I_2} \Box(p_{2,j} \implies \bigcirc q_{2,j}),$$
$$\varphi_{per} := \Diamond \Box p_3, \qquad \varphi_{rec} := \bigwedge_{j \in I_4} \Box \Diamond p_{4,j}$$

and $p_1 := \bigwedge_{j \in I_1} p_{1,j}$ and $p_3 := \bigwedge_{j \in I_3} p_{3,j}$ with $\bigwedge_{j \in I_1} \Box p_{1,j} = \Box \bigwedge_{j \in I_1} p_{1,j}$ and $\bigwedge_{j \in I_3} \Diamond \Box p_{3,j} = \Diamond \Box \bigwedge_{j \in I_3} p_{3,j}$, respectively. In the above definitions, $I_1, \ldots, I_4$ are finite index sets and $p_{i,j}$ and $q_{i,j}$ are propositional formulas for any i and j.

Remark 1. Guarantee and obligation, i.e., $\Diamond p$ and $\Box(p \implies \Diamond q)$ respectively (where p and q are propositional formulas), are not included in (1). We show how to include these specifications in Section VIII-A. It is also natural to consider specifications that are disjunctions of formulas of the form (1). We give conditions for this extension in Section VIII-B.

Remark 2. It is clear that the fragment in formula (1) is a strict subset of LTL. This fragment is incomparable to other commonly used temporal logics, such as computational tree logic (CTL) and generalized reactivity(1) (GR(1)). The fragment that we consider allows persistence ($\Diamond\Box$) properties to be specified, which cannot be specified in either CTL or GR(1). However, it cannot express existential path quantification as in CTL or allow disjunctions of formulas as in GR(1) [2], [5]. The fragment is a subset of the generalized Rabin(1) logic [9] and the μ-calculus of alternation depth two [11].

III. Preliminaries

A. Acceptance conditions

We now give acceptance conditions from classical automata theory [13] for the LTL fragment introduced in Section II-D. These acceptance conditions are critical to the development in this paper, as much of the later analysis depends on them. Effectively, we reason about satisfaction of LTL formulas in terms of set operations between a run σ of $\mathcal{T}$ and subsets of S. These sets correspond to states where particular propositional formulas hold. We first define acceptance conditions for a run and then extend it to a system $\mathcal{T}$.

Definition 2. Let σ be a run of the system $\mathcal{T}$, let *Inf*(σ) denote the set of states that are visited infinitely often in σ, and let *Vis*(σ) denote the set of states that are visited at least once in σ. Given propositional formulas φ and ψ, we relate satisfaction of an LTL formula with acceptance conditions as follows

- $\sigma \vDash \Box\varphi$ iff *Vis*$(\sigma) \subseteq [\varphi]$,
- $\sigma \vDash \Diamond \Box \varphi$ iff *Inf*$(\sigma) \subseteq [\varphi]$,
- $\sigma \vDash \Box \Diamond \varphi$ iff *Inf*$(\sigma) \cap [\varphi] \neq \varnothing$,
- $\sigma \vDash \Box(\psi \implies \bigcirc\varphi)$ iff $\sigma_i \in [\psi]$ or $\sigma_{i+1} \notin [\varphi]$ for all i.

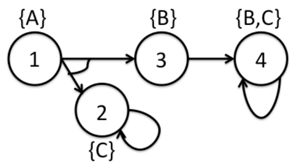

Fig. 1. Example of a non-deterministic transition system

A run satisfies a conjunction of LTL formulas if and only if it satisfies all corresponding acceptance conditions. Acceptance for a system $\mathcal{T}$ is extended over runs in the obvious manner.

In automata theory, $\Box \Diamond \varphi$ is called a Büchi acceptance condition and $\Diamond\Box\varphi$ is called a co-Büchi acceptance condition. The conjunction of both a Büchi and a co-Büchi acceptance condition is a Rabin acceptance condition with one pair [13].

An example is given in Figure 1. The non-deterministic transition system $\mathcal{T}$ has states $S = \{1,2,3,4\}$; labels $L(1) = A$, $L(2) = C$, $L(3) = B$, $L(4) = B, C$; a single action called 0; and transitions $R(1,0) = \{2,3\}$, $R(2,0) = \{2\}$, $R(3,0) = \{4\}$, $R(4,0) = \{4\}$. Using the acceptance conditions, it follows that states $\{2,4\}$ are winning for formula $\Box(A \vee C)$, states $\{2,3,4\}$ are winning for formula $\Box(A \implies \bigcirc B)$, states $\{1,2,3,4\}$ are winning for formula $\Box \Diamond C$, and states $\{3,4\}$ are winning for formula $\Diamond \Box B$. State 4 is the only state that is winning for all of the formulas above.

B. Graph Theory

We will often consider a non-deterministic transition system as a graph with the natural bijection between the states and transitions of the transition system and the vertices and edges of the graph. Let $G = (S, R)$ be a directed graph (digraph) with vertices S and edges R. Let there be an edge e from vertex s to vertex t if and only if $t \in R(s,a)$ for some $a \in A(s)$. A *walk* w is a finite edge sequence $w = e_0e_1\ldots e_p$. Denote the set of all nodes visited along walk w by *Vis*(w).

A digraph $G = (S, R)$ is *strongly connected* if there exists a path between each pair of vertices $s, t \in S$ no matter how the environment resolves the non-determinism. A digraph $G' = (S', R')$ is a *subgraph* of $G = (S, R)$ if $S' \subseteq S$ and $R' \subseteq R$. The subgraph of G restricted to states $S' \subseteq S$ is denoted by $G|_{S'}$. A digraph $G' \subseteq G$ is a *strongly connected component* if it is a maximal strongly connected subgraph of G.

C. Reachability

We define *controlled reachability* in a non-deterministic transition system $\mathcal{T}$ with a value function. Let $B \subseteq S$ be a set of states that the controller wants the system to reach. Let the *controlled value function* for system $\mathcal{T}$ and target set B be a map $V^c_{B,\mathcal{T}} : S \to \mathbb{N} \cup \infty$, whose value $V^c_{B,\mathcal{T}}(s)$ at state $s \in S$ is the minimum (over all possible control policies) number of transitions needed to reach the set B, given the worst-case resolution of the non-determinism. If the value $V^c_{B,\mathcal{T}}(s) = \infty$, then the non-determinism can prevent the system from reaching set B from state $s \in S$. For example, consider the system in Figure 1 with $B = \{4\}$. Then, $V^c_B(1) = \infty$, $V^c_B(2) = \infty$, $V^c_B(3) = 1$, and $V^c_B(4) = 0$.

The value function satisfies the optimality condition

$$V^c_{B,\mathcal{T}}(s) = \min_{a \in A(s)} \max_{t \in R(s,a)} V^c_{B,\mathcal{T}}(t), \tag{2}$$

for all $s \in S$. Algorithm 1 computes the value function by backwards iteration from the target set B. At every iteration, a state is assigned a finite value if it can reach a state with a finite value in a single transition, no matter how the non-determinism is resolved. It is easy to see that the recursion takes $O(|S| + |R|)$ time.

Algorithm 1 Value function (controlled)

```
Input: NTS $\mathcal{T}$, set $B \subseteq S$
Output: The (controlled) value function $V^c_{B,\mathcal{T}}$
  $V^c_{B,\mathcal{T}}(s) \leftarrow 0$ for all $s \in B$; $V^c_{B,\mathcal{T}}(s) \leftarrow \infty$ for all $s \in S - B$
  while $B \neq \varnothing$ do
    $C \leftarrow \varnothing$
    for $\{s \in Parents(B) \mid V^c_{B,\mathcal{T}}(s) = \infty\}$ do
      $V^c_{B,\mathcal{T}}(s) \leftarrow \min_{a \in A(s)} \max_{t \in R(s,a)} V^c_{B,\mathcal{T}}(t)$
      if $V^c_{B,\mathcal{T}}(s) < \infty$ then
        $C \leftarrow C \cup \{s\}$
    $B \leftarrow C$
  return $V^c_{B,\mathcal{T}}$
```

An optimal control policy μ_B for reaching the set B is implicitly encoded in a value function $V^c_{B,\mathcal{T}}$ that satisfies (2). Optimal control policies are memoryless for reachability [4]. Such a policy can be computed at each state $s \in S$ as

$$\mu_B(s) = \arg\min_{a \in A(s)} \max_{t \in R(s,a)} V^c_{B,\mathcal{T}}(t). \tag{3}$$

We use the value function to define the *controllable predecessor* set for a given system $\mathcal{T}$ with target set $B \subseteq S$. Let

$$CPre^{\infty}_{\mathcal{T}}(B) \coloneqq \{s \in S \mid V^c_{B,\mathcal{T}}(s) < \infty\} \tag{4}$$

be the set of all states that can reach a state in B for any resolution of the non-determinism.

We define *forced reachability* similarly. Let the *forced value function* for system $\mathcal{T}$ and target set B be a map $V^f_{B,\mathcal{T}} : S \to \mathbb{N} \cup \infty$, whose value $V^f_{B,\mathcal{T}}(s)$ at state $s \in S$ is the maximum (over all possible control policies) number of transitions before reaching the set B. The forced value function satisfies the optimality condition

$$V^f_{B,\mathcal{T}}(s) = \max_{a \in A(s)} \max_{t \in R(s,a)} V^f_{B,\mathcal{T}}(t). \tag{5}$$

For a given system $\mathcal{T}$ with target set $B \subseteq S$, the *forced predecessor* set

$$FPre^{\infty}_{\mathcal{T}}(B) \coloneqq \{s \in S \mid V^f_{B,\mathcal{T}}(s) < \infty\}, \tag{6}$$

is the set of all states from which no control policy can avoid reaching a state in B.

Remark 3. We consider the case where the controller selects an action, and then the environment selects the next state. Our results are easily extended to the case where the environment first selects a state for each possible action, and then the controller selects an action.

IV. Solution for deterministic transition systems

We first discuss algorithms for computing control policies for deterministic transition systems, as these are conceptually simpler than their non-deterministic generalization. We will compute the winning set $W \subseteq S$ for each specification separately and then combine them in Algorithm 2. We remind the reader that $\mathcal{T}$ is originally non-blocking.

We first remove all actions from $\mathcal{T}$ that do not satisfy the next-step response specification $\varphi_{act} = \bigwedge_{j \in I_2} \square(p_{2,j} \implies \bigcirc q_{2,j})$. For each $j \in I_2$, remove an action $a \in A(s)$ from a state $s \in S$ if $s \in [p_{2,j}]$ and $R(s,a) \notin [q_{2,j}]$. Let $B \subseteq S$ contain all states that are blocking (due to the removal of an action). Create the subgraph $\mathcal{T}_{act} \coloneqq \mathcal{T}|_{S - FPre^{\infty}_{\mathcal{T}}(B)}$.

Proposition 1. *A state is in $\mathcal{T}_{act}$ if and only if it is winning for φ_{act}.*

Proof: An action is removed from $\mathcal{T}$ if and only if it directly violates acceptance condition for φ_{act}. All blocking states, i.e., those in $B \subseteq S$, must use an action that was removed. Thus, the set $FPre^{\infty}_{\mathcal{T}}(B)$ contains all and only states that must violate the acceptance condition for φ_{act}. $\mathcal{T}_{act}$ is non-blocking, so any run of the system satisfies φ_{act}. ■

We next remove the states that violate the safety specification $\varphi_{safe} = \square p_1$ by creating the subgraph $\mathcal{T}_{safe} \coloneqq \mathcal{T}|_{S - FPre^{\infty}_{\mathcal{T}}(S - [p_1])}$.

Proposition 2. *A state is in $\mathcal{T}_{safe}$ if and only if it is winning for φ_{safe}.*

Proof: The acceptance condition for φ_{safe} is *Vis*$(\sigma) \subseteq [p_1]$. The set $FPre^{\infty}_{\mathcal{T}}(S - [p_1]))$ contains a state if and only if it either is not in $[p_1]$ and or cannot avoid visiting a state not in $[p_1]$. $\mathcal{T}_{safe}$ is non-blocking, so any run of the system satisfies φ_{safe}. ■

We incorporate the persistence specification $\varphi_{per} = \diamond \square p_3$ by creating the subgraph $\mathcal{T}_{per} \coloneqq \mathcal{T}|_{S - FPre^{\infty}_{\mathcal{T}}(S - [p_3])}$. The winning set is $CPre^{\infty}_{\mathcal{T}}(S_{per})$, where S_{per} is the set of states in $\mathcal{T}_{per}$.

Proposition 3. *A state is in $\mathcal{T}_{per}$ if it is winning for φ_{per}.*

Proof: As in Proposition 2, but with acceptance condition *Inf*$(\sigma) \subseteq [p_3]$. ■

We now compute the winning set for the recurrence specification $\varphi_{rec} = \bigwedge_{j \in I_4} \square \diamond p_{4,j}$ by computing the sets of states that can be visited infinitely often.

Proposition 4. *Let σ be a run of $\mathcal{T}$. If states $s, t \in Inf(\sigma)$, then they must be in the same strongly connected component.*

Proof: By definition of *Inf*(σ), states s and t are visited infinitely often. Thus, there must exist a walk starting at s

and ending at t and vice versa. Thus, s and t are in the same strongly connected component. ■

The strongly connected components of $\mathcal{T}$ can be computed in $O(|S| + |R|)$ time using Tarjan's algorithm [6]. Let $SCC(\mathcal{T}_{per})$ be the set of all strongly connected components of $\mathcal{T}_{per}$ that have at least one transition between states in the component. A strongly connected component $C \in SCC(\mathcal{T}_{per})$ is *accepting* if $C \cap [p_{4,j}] \neq \varnothing$ for all $j \in I_4$. Let $\mathcal{A}$ be the set of all accepting strongly connected components and $S_{\mathcal{A}} := \{s \in S \mid s \in C$ for some $C \in \mathcal{A}\}$. Every state in an accepting strongly connected component is in the winning set $W := CPre^{\infty}_{\mathcal{T}}(S_{\mathcal{A}})$.

Proposition 5. *A state is in $S_{\mathcal{A}}$ if it is winning for φ_{rec}.*

Proof: The relevant acceptance condition is *Inf*$(\sigma) \cap [p_{4,j}] \neq \varnothing$ for all $j \in I_4$. By definition, every state in $C \in \mathcal{A}$ can be visited infinitely often. Since $C \cap [p_{4,j}] \neq \varnothing$ for all $j \in I_4$, the result follows. ■

We now give an overview of our approach for control policy synthesis for deterministic transition systems in Algorithm 2. Optionally, remove all states from $\mathcal{T}$ that cannot be reached from the initial state s_0 in $O(|S| + |R|)$ time using breadth-first search from s_0 [6]. Compute the set of states W that are winning for φ (lines 1-4). If the initial state $s_0 \notin W$, then no control policy exists (lines 5-6). If $s_0 \in W$, compute a walk on $\mathcal{T}_{safe}$ from s_0 to a state $t \in C$ for some accepting strongly connected component $C \in \mathcal{A}$ and where $t \in \bigcup_{j \in I_4}[p_{4,j}]$ (lines 8-9). Compute a walk σ_{suf} starting and ending at state t such that *Vis*$(\sigma_{suf}) \cap [p_{4,j}] \neq \varnothing$ for all $j \in I_4$ and *Vis*$(\sigma_{suf}) \subseteq C$ (line 10). The control policy is implicit in the (deterministic) run $\sigma = \sigma_{pre}(\sigma_{suf})^{\omega}$, where ω denotes infinite repetition. The total complexity of the algorithm is $O(|I_2|(|S| + |R|))$.

Algorithm 2 Overview: Synthesis for DTS

```
Input: DTS $\mathcal{T}$, $s_0 \in S$, formula $\varphi$
Output: Run $\sigma$
 1: $\mathcal{T}_{safe} \leftarrow \mathcal{T}_{act}|_{S-FPre^{\infty}_{\mathcal{T}_{act}}(S-[p_1])}$
 2: $\mathcal{T}_{per} \leftarrow \mathcal{T}_{safe}|_{S-FPre^{\infty}_{\mathcal{T}_{safe}}(S-[p_3])}$
 3: $\mathcal{A} := \{C \in$ SCC $(\mathcal{T}_{per}) \mid C \cap [p_{4,j}] \neq \varnothing \; \forall j \in I_4\}$
 4: $S_{\mathcal{A}} := \{s \in S \mid s \in C$ for some $C \in \mathcal{A}\}$
 5: if $s_0 \notin W := CPre^{\infty}_{\mathcal{T}_{safe}}(S_{\mathcal{A}})$ then
 6:     return "no satisfying control policy exists"
 8: Pick state $t \in C$ for some $C \in \mathcal{A}$ and $t \in \bigcap_{j \in I_4}[p_{4,j}]$
 9: Compute walk $\sigma_{pre}$ from $s_0$ to $t$ s.t. Vis$(\sigma_{pre}) \subseteq W$
10: Compute walk $\sigma_{suf}$ from $t$ to $t$, s.t. Vis$(\sigma_{suf}) \subseteq C \subseteq W$
    and Vis$(\sigma_{suf}) \cap [p_{4,j}] \neq \varnothing \; \forall j \in I_4$
11: return $\sigma = \sigma_{pre}(\sigma_{suf})^{\omega}$
```

V. Non-deterministic transition system

We now discuss control policy construction for non-deterministic transition systems, which subsumes the development in Section IV. Our approach here differs primarily in the form of the control policy and how we determine the set of states that satisfy a recurrence formula.

We address formulas for next-step response φ_{act}, safety φ_{safe}, and persistence φ_{per} in a similar manner as Section IV because both the set $FPre^{\infty}$ and the subgraph operation are already defined for non-deterministic transition systems.

Next, we consider the recurrence specification $\varphi_{rec} = \bigwedge_{j \in I_4} \square \diamond p_{4,j}$. A similar approach to the strongly connected component decomposition in Section IV could be used, but it is less efficient to compute such a decomposition due to the non-determinism. Büchi (and the more general parity) acceptance conditions have been extensively studied [5], [13].

Proposition 6. *Algorithm 3 computes the winning set for φ_{rec}.*

Proof: To satisfy the acceptance condition *Inf*$(\sigma) \cap [p_{4,j}] \neq \varnothing$ for all $j \in I_4$, $F_i \subseteq CPre^{\infty}_{\mathcal{T}}(F_j)$ must hold for all $i, j \in I_4$ for some $F_j \subseteq [p_{4,j}]$. Algorithm 3 initializes $F_j := [p_{4,j}]$ for all $j \in I_4$ and iteratively removes states from F_i that are not in $CPre^{\infty}_{\mathcal{T}}(F_j)$ for all $i, j \in I_4$. It terminates only when $F_i \subseteq CPre^{\infty}_{\mathcal{T}}(F_j)$ holds for all $i, j \in I_4$ or $F_i = \varnothing$ for some $i \in I_4$, i.e., the winning set is empty. If it does not terminate during an iteration, it must remove at least one state in $F = \bigcup_{j \in I_4} F_j$. ■

Algorithm 3 BUCHI $(\mathcal{T}, \{[p_{4,1}], \ldots, [p_{4,|I_4|}]\})$

```
Input: NTS $\mathcal{T}$, $[p_{4,j}] \subseteq S$ for $j \in I_4$
Output: Winning set $W \subseteq S$
  $F_j := [p_{4,j}]$ for all $j \in I_4$; update ← True
  while update do
    update ← False
    for $i \in I_4$ do
      for $j \in I_4$ do
        if $F_j \nsubseteq CPre^{\infty}_{\mathcal{T}}(F_i)$ then
          update ← True
        $F_j \leftarrow F_j \cap CPre^{\infty}_{\mathcal{T}}(F_i)$
        if $F_j = \varnothing$ then
          return $W \leftarrow \varnothing$, $F_j$ for all $j \in I_4$
  return $W \leftarrow CPre^{\infty}_{\mathcal{T}}(F_1)$, $F_j$ for all $j \in I_4$
```

We now overview our approach for control policy synthesis for non-deterministic transition systems in Algorithm 4. Compute the set W of states that are winning for φ (lines 1-5). If the initial state $s_0 \notin W$, then no control policy exists. If $s_0 \in W$, compute the memoryless control policies μ_j induced from $V^c_{F_j, \mathcal{T}_{safe}}$ for all $j \in I_4$ (line 9, also see Algorithm 3). The finite-memory control policy μ is defined as follows by switching between memoryless policies depending on the current "target." Let $j \in I_4$ denote the current target set F_j. The system uses control policy μ_j until a state in F_j is visited. Then, the system updates its "target" to $k = (j+1 \mod |I_4|)+1$ and uses control policy μ_k until a state in F_k is visited, and so on. The total complexity of the algorithm is $O(n|F|(|S|+|R|))$, where $n = \max\{|I_2|, |I_4|\}$ and $F = \bigcup_{j \in I_4}[p_{4,j}]$.

VI. Markov Decision Processes

We now consider the Markov decision process (MDP) model. MDPs provide a general framework for modeling non-determinism and probabilistic behaviors that are present in

Algorithm 4 Overview: Synthesis for NTS

```
Input: Non-deterministic TS $\mathcal{T}$ and formula $\varphi$
Output: Control policy $\mu$
 1: $\mathcal{T}_{safe} \leftarrow \mathcal{T}_{act}|_{S-FPre^{\infty}(S-[p_1])}$
 2: $\mathcal{T}_{per} \leftarrow \mathcal{T}_{safe}|_{S-FPre^{\infty}(S-[p_3])}$
 3: $P := \{[p_{4,1}], \ldots, [p_{4,|I_4|}]\}$
 4: $S_{\mathcal{A}}, F := \{F_1, \ldots, F_{|I_4|}\} \leftarrow$ BUCHI$(\mathcal{T}_{per}, P)$
 5: $W := CPre^{\infty}_{\mathcal{T}_{safe}}(S_{\mathcal{A}})$
 6: if $s_0 \notin W$ then
 8:     return "no satisfying control policy exists"
 9: $\mu_j \leftarrow$ control policy induced by $V^c_{F_j,\mathcal{T}_{safe}}$ for all $j \in I_4$
10: return $\{\mu_1, \ldots, \mu_{|I_4|}\}$ {set of control policies}
```

many real-world systems. We sketch an approach for control policy synthesis for formulas of the form $\varphi = \varphi_{safe} \wedge \varphi_{per} \wedge \varphi_{rec}$ using techniques from probabilistic model checking [2]. This terse presentation will be extended in later publications.

Definition 3. A (finite) *labeled MDP* $\mathcal{M}$ is the tuple $\mathcal{M} = (S, A, P, s_0, AP, L)$, consisting of a finite set of states S, a finite set of actions A, a transition probability function $P : S \times A \times S \to [0, 1]$, an initial state s_0, a finite set of atomic propositions AP, and a labeling function $L : S \to 2^{AP}$. Let $A(s)$ denote the set of available actions at state s. Let $\sum_{s' \in S} P(s, a, s') = 1$ if $a \in A(s)$ and $P(s, a, s') = 0$ otherwise. We assume, for notational convenience, that the available actions $A(s)$ are the same for every $s \in S$.

A *run* of the MDP is an infinite sequence of its states, $\sigma = s_0 s_1 s_2 \ldots$ where $s_i \in S$ is the state of the system at index i and $P(s_i, a, s_{i+1}) > 0$ for some $a \in A(s_i)$. The set of runs of $\mathcal{M}$ with initial state s induced by a control policy μ (as defined in Section II-A) is denoted by $\mathcal{M}^{\mu}(s)$. There is a probability measure over the runs in $\mathcal{M}^{\mu}(s)$ [2].

Given a run of $\mathcal{M}$, the syntax and semantics of LTL is identical to Section II-B. However, satisfaction for an MDP $\mathcal{M}$ under a control policy μ is now defined probabilistically [2]. Let $\mathbb{P}(M^{\mu}(s) \vDash \varphi)$ denote the *expected satisfaction probability* of LTL formula φ by $\mathcal{M}^{\mu}(s)$.

Problem 2. Given an MDP $\mathcal{M}$ with initial state s_0 and an LTL formula φ, compute the optimal control policy $\mu^* = \arg\max_{\mu} \mathbb{P}(\mathcal{M}^{\mu}(s_0) \vDash \varphi)$, over all possible finite-memory, deterministic policies.

The value function at a state now has the interpretation as the maximum probability of the system satisfying the specification from that state. Let $B \subseteq S$ be a set from which the system can satisfy the specification almost surely. The value $V_{B,\mathcal{M}}(s)$ of a state $s \in S$ is the probability that the MDP $\mathcal{M}$ will reach set $B \subseteq S$ when using an optimal control policy starting from state $s \in S$.

We first compute the winning set $W \subseteq S$ for the LTL formula $\varphi = \varphi_{safe} \wedge \varphi_{per} \wedge \varphi_{rec}$. The probability of satisfying φ is equivalent to the probability of reaching an *accepting maximal end component* [2]. Informally, accepting maximal end components are sets of states that the system can remain in forever and where the acceptance condition of φ is satisfied almost surely. These sets can be computed in $O(|S||R|)$ time using graph search [2], similar to the approach used in Section V. The winning set $W \subseteq S$ is the union of all states that are in some accepting maximal end component.

TABLE I
COMPLEXITY OF CONTROL POLICY SYNTHESIS

Language	DTS	NTS	MDP
Fragment in (1)	$O(n\|\mathcal{T}\|)$	$O(n\|F\|\|\mathcal{T}\|)$	$O(poly(\|\mathcal{T}\|))$
GR(1)	$O(mn\|S\|\|R\|)$	$O(mn\|S\|\|R\|)$	N/A
LTL	$O(\|\mathcal{T}\|2^{(\|\varphi\|)})$	$O(\|\mathcal{T}\|2^{2^{(\|\varphi\|)}})$	$O(poly(\|\mathcal{T}\|)2^{2^{(\|\varphi\|)}})$

A. Reachability

Once we have computed the winning set $W \subseteq S$ where the system can satisfy the specification φ almost surely, we need to reach W from the initial state s_0. Let $\mathcal{M}_{safe}$ be the sub-MDP (defined similarly to Section III-B, see [2]) where all states satisfy φ_{safe}. The set $S_1 = CPre^{\infty}(W)$ contains all states that can satisfy φ almost surely. Let S_r be the set of states that have positive probability of reaching W, which can be computed by graph search [2]. The remaining states $S_0 = S - (S_1 \cup S_r)$ cannot reach W and thus have zero probability of satisfying φ. Initialize $V^c_B(s) = 1$ for all $s \in S_1$, $V^c_B(s) = 0$ for all $s \in S_0$, and $V^c_B(s) \in (0, 1)$ for all $s \in S_r$. It remains to compute the value function, i.e. the maximum probability of satisfying the specification, for each state in S_r. This computation boils down to a standard reachability problem that can be solved by linear programming or value iteration. [2], [4].

B. Control policy

The optimal control policy for satisfying the LTL formula φ consists of two parts: a memoryless deterministic policy for reaching an accepting maximal end component, and a finite-memory deterministic policy for staying there. The former policy is computed from $V^c_{B,\mathcal{M}}$ and denoted μ_{reach}. The latter policy is a finite-memory policy μ_B that selects actions to ensure that the system stays inside the accepting maximal end component forever and satisfies φ by visiting every state infinitely often [2]. The optimal control policy μ^* is $\mu^* = \mu_{reach}$ if $s \notin B$ and $\mu^* = \mu_B$ if $s \in B$.

VII. COMPLEXITY

We summarize our complexity results for control policy synthesis for deterministic transition systems, non-deterministic transition systems, and MDPs. We also compare our results with those for LTL and the commonly used GR(1) fragment of LTL [5]. Let $|\mathcal{T}| = |S| + |R|$ denote the size of the system, $n = \max\{|I_2|, |I_4|\}$, and $F = \bigcup_{j \in I_4}[p_{4,j}]$. We use $poly(|\mathcal{T}|)$ to denote that the complexity is polynomial in $|\mathcal{T}|$, specifically that of solving a linear program. Note that for typical motion planning specifications, F is much smaller than S and n is small. For a GR(1) formula, m and n are the number of assumptions (N/A for fragment (1)) and guarantees ($|I_4|$ for fragment (1)) respectively, and we use the non-symbolic complexity results [5]. Results are summarized in Table I.

VIII. Extensions

We now discuss two natural extensions to the fragment in formula (1). The first is specifying guarantee and obligation properties and the second is including disjunctions of formulas.

A. Guarantee and obligation

While guarantee and obligation, i.e., $\Diamond p$ and $\Box(p \implies \Diamond q)$ respectively (where p and q are propositional formulas), specifications are not explicitly included in (1), they can be incorporated by introducing new system variables. An example of this approach is given in [25]. The size of the system grows exponentially in the number of additional variables used. In fact, control policy synthesis for conjunctions of guarantee formulas is NP-complete [24].

Another approach is to use the stricter specifications $\Box \Diamond p$ for guarantee and $\Box \neg p \lor \Box \Diamond q$ for obligation. The $\Box\Diamond$ formulas are part of the fragment in (1), and disjunctions can be included in some cases (see Section VIII-B). If the transition system is strongly connected, then these stricter formulas are feasible if and only if the original formulas are because all states in a strongly connected component can be visited infinitely often. Strong connectivity is a natural assumption in many motion planning applications. For example, an autonomous car can typically drive around the block to revisit a location.

B. Disjunctions of specifications

We now consider an extension to specifications that are disjunctions of formulas of the form (1).

For a deterministic transition system, a control policy for a formula given by disjunctions of formulas of the form (1) can be computed by independently solving each individual subformula using the algorithms given earlier in this section.

Proposition 7. *Let $\varphi = \varphi_1 \lor \varphi_2 \lor \ldots \lor \varphi_n$ where φ_i is a formula of the form* (1) *for $i = 1, \ldots, n$. Then, there exits a control policy μ such that $\mathcal{T}^\mu(s) \models \varphi$ if and only if there exists a control policy μ such that $\mathcal{T}^\mu(s) \models \varphi_i$ for some $i = 1, \ldots, n$.*

Proof: Sufficiency is obvious. For necessity, assume that there exists a control policy μ such that $\mathcal{T}^\mu(s)$ satisfies φ. The set $\mathcal{T}^\mu(s)$ contains a single run σ since $\mathcal{T}$ is deterministic. Thus, σ satisfies φ_i for some $i = 1, \ldots, n$. ∎

For non-deterministic transition systems, necessity in Proposition 7 no longer holds because the non-determinism may be resolved in multiple ways and thus independently evaluating each subformula may not work.

Algorithm 5 is a sound, but not complete, procedure for synthesizing a control policy a non-deterministic transition with a specification given by disjunctions of formulas of the form (1). Arbitrary disjunctions of this form cannot be solved both efficiently and exactly, as this extension subsumes Rabin games which are NP-complete [10]. Future work will determine under what conditions disjunctions can be incorporated.

Algorithm 5 independently computes winning sets $W_i \subseteq S$ for each subformula φ_i and checks if the initial state can reach their union $\mathcal{W} := \bigcup_{i=1}^n W_i$. The control policy μ_{reach} is then used until a state $s \in W_i$ is reached for some i. Then, μ_i is used. We are investigating whether or not this approach can be done recursively.

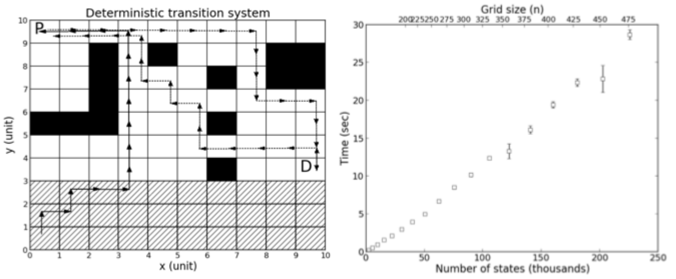

Fig. 2. Left: A diagram of a 10 x 10 grid. Only white cells are labeled 'stockroom.' Right: Control policy synthesis times for five random grids.

Algorithm 5 DISJUNCTION

```
Input: NTS $\mathcal{T}$, formula $\varphi_i$, $i = 1, \ldots, n$
Output: Winning set $W \subseteq S$ and control policy $\mu$
  $W_i \subseteq S$ and $\mu_i \leftarrow$ winning states and control policy for $\varphi_i$
  $\mathcal{W} \leftarrow \bigcup_{i=1}^n W_i$
  if $s_0 \notin CPre_{\mathcal{T}}^{\infty}(\mathcal{W})$ then
      return $\mu = \varnothing$
  $\mu_{reach} \leftarrow$ control policy induced by $V_{\mathcal{W},\mathcal{T}}^c$
  return $\mu_{reach}$ and $\mu_i$ for all $i$
```

IX. Examples

The following examples demonstrate the techniques developed in Sections IV and V for tasks motivated by robot motion planning in a planar environment. We defer an example for Section VI due to space limitations. The software was written in Python and computations were done on a Linux desktop with a dual-core processor and 2 GB of memory. All computation times were averaged over five randomly generated problem instances.

A. Deterministic transition system

Consider a gridworld where a robot occupies a single cell at a time and can choose to either remain in its current cell or move to one of four adjacent cells at each step. We consider square grids with static obstacle densities of 15 percent. The set of atomic propositions is $AP = \{\text{pickup}, \text{dropoff}, \text{storeroom}, \text{obs}\}$. The robot's task is to eventually remain in the stockroom while repeatedly visiting a pickup and a dropoff location. The robot must never collide with a static obstacle. This task is formalized by the LTL formula $\varphi = \Diamond\Box\text{stockroom} \land \Box\Diamond\text{pickup} \land \Box\Diamond\text{dropoff} \land \Box\neg\text{obs}$, which is in fragment (1).

Results are shown in Figure 2. A corresponding non-deterministic Büchi automaton for φ has four states [12]. Thus, the standard automata-based approach for LTL would do similar graph search computations on a graph four times larger than the transition system.

B. Non-deterministic transition system

We now consider a similar setup as in Section IX-A, except that there is now a dynamically moving obstacle. The state

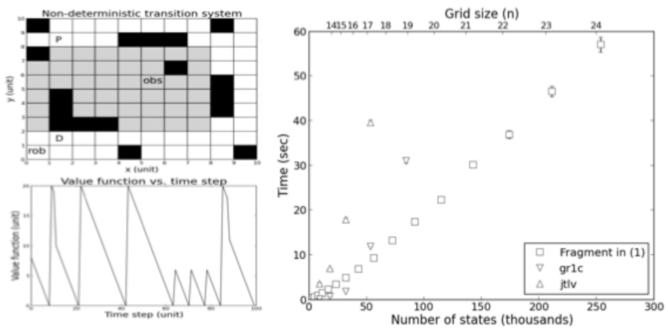

Fig. 3. Upper left: A diagram of a 10 x 10 grid with a dynamic obstacle that moves within the shaded region. Lower left: The value function at the current state. Downward jumps indicate the robot exploits a non-optimal move by the obstacle. Right: Control policy synthesis times for five random grids.

of the system is the product of the robot's location and the obstacle's location, both of which can move as previously described for the robot. The robot selects an action and then the obstacle non-deterministically moves. The robot's task is to repeatedly visit a pickup and a dropoff location while never colliding with an obstacle. This task is formalized by the LTL formula $\varphi = \square \diamond \text{pickup} \wedge \square \diamond \text{dropoff} \wedge \square \neg \text{obs}$, which is in both fragment (1) and generalized reactivity(1) [5].

Results are shown in Figure 3. We compare our algorithm to two implementations (jtlv and gr1c as used in [26]) of the generalized reactivity(1) synthesis method from [5]. Our algorithms scale significantly better; neither the jtlv or gr1c implementation was able to solve a problem with over 100 thousand states. Finally, to highlight the reactive nature of our control policy, we show the value function as it varies along a simulated run where the obstacle moves randomly.

X. Conclusions

We presented a framework for control policy synthesis for both non-deterministic transition systems and Markov decision processes that are subject to temporal logic task specifications. Our approach for control policy synthesis is straight-forward and efficient, both theoretically and according to our preliminary experimental results. It offers a promising alternative to the commonly used generalized reactivity(1) specifications as it can express many relevant tasks for multiple system models.

Future work will extend the synthesis algorithms here to create optimal control policies for systems with appropriately defined costs. Incremental synthesis methods for computing the reachable sets also appear promising. Finally, while this fragment appears promising compared to generalized reactivity(1) and LTL, detailed experimental analysis is needed.

Acknowledgements

The authors would like to thank Scott Livingston for help with the software implementation. This work was supported by a NDSEG fellowship, the Boeing Corporation, and AFOSR award FA9550-12-1-0302.

References

[1] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Logic*, 5(1):1–25, 2004.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[3] C. Belta and L. Habets. Control of a class of non-linear systems on rectangles. *IEEE Transactions on Automatic Control*, 51:1749–1759, 2006.

[4] D. P. Bertsekas. *Dynamic Programming and Optimal Control (Vol. I and II)*. Athena Scientific, 2001.

[5] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) designs. *Journal of computer and system sciences*, 78:911–938, 2012.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms: 2nd edition*. MIT Press, 2001.

[7] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the Association for Computing Machinery*, 42:857–907, 1995.

[8] X. C. Ding, S. L. Smith, C. Belta, and D. Rus. LTL control in uncertain environments with probabilistic satisfaction guarantees. In *Proceedings of 18th IFAC World Congress*, 2011.

[9] R. Ehlers. Generalized Rabin(1) synthesis with applications to robust system synthesis. In *NASA Formal Methods*. Springer, 2011.

[10] E. Emerson and C. Jutla. The complexity of tree automata and logic of programs. In *In 29th FOCS*, 1988.

[11] E. A. Emerson. Handbook of theoretical computer science (vol. B). In J. van Leeuwen, editor, *Temporal and modal logic*, chapter Temporal and modal logic, pages 995–1072. MIT Press, 1990.

[12] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Proceedings of the 13th International Conference on Computer Aided Verification*, 2001.

[13] E. Gradel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag New York, Inc., 2002.

[14] L. Habets, P. Collins, and J. van Schuppen. Reachability and control synthesis for piecewise-affine hybrid systems on simplices. *IEEE Transaction on Automatic Control*, 51:938–948, 2006.

[15] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Proc. of IEEE Conference on Decision and Control*, 2009.

[16] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

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

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

[19] S. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20:378–400, 2001.

[20] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 95*, volume 900, pages 229–242. Springer, 1995.

[21] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *Proc. International Conference on Verification, Model Checking and Abstract Interpretation*, pages 364 – 380, 2006. Software available at http://jtlv.sourceforge.net/.

[22] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Transactions on Robotics*, 26:469–482, 2010.

[23] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. Symposium on Principles of Programming Languages*, pages 179–190, 1989.

[24] A. Sistla and E. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32:733–749, 1985.

[25] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 2012. (to appear).

[26] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. TuLiP: A software toolbox for receding horizon temporal logic planning. In *International Conference on Hybrid Systems: Computation and Control*, 2011. http://tulip-control.sf.net.