Submitted, 2015 International Conference on Hybrid Systems: Computation and Control (HSCC)
http://www.cds.caltech.edu/~murray/papers/rfwm15-hscc.html

Online Horizon Selection in Receding Horizon Temporal Logic Planning

Vasumathi Raman
California Institute of Technology
Pasadena, CA, USA
vasu@caltech.edu

Mattias Fält
Lund University
Lund, Sweden
faldt.mattias@gmail.com

Tichakorn Wongpiromsarn
Thailand Center of Excellence for Life Sciences
Bangkok, Thailand
tichakorn@tcels.or.th

Richard M. Murray
California Institute of Technology
Pasadena, CA, USA
murray@cds.caltech.edu

ABSTRACT

Temporal logics have proven effective for correct-by-construction synthesis of controllers for a wide range of applications. Receding horizon frameworks mitigate the computational intractability of reactive synthesis for temporal logic, but have thus far been limited by pursuing a single sequence of short horizon problems to the current goal. We propose a receding horizon algorithm for reactive synthesis that automatically determines a path to the currently pursued goal at runtime, in response to a nondeterministic environment. This is achieved by allowing each short horizon to have multiple local goals, and determining which local goal to pursue based on the current global goal, currently perceived environment and a pre-computed invariant dependent on each global goal. We demonstrate the utility of this additional flexibility in grant-response tasks, using a search-and-rescue example. Moreover, we show that these goal-dependent invariants mitigate the conservativeness of the receding horizon approach.

1. INTRODUCTION

Temporal logics have proved an effective formalism for specifying, verifying and synthesizing behaviors of a variety of hybrid systems. Algorithms for temporal logic synthesis enable automated construction of discrete supervisory controllers satisfying intricate temporal sequencing properties; these discrete controllers have been successfully used to construct hybrid controllers for several domains including robotics [6, 8], aircraft power system design [13] and smart buildings [15].

Linear Temporal Logic (LTL) has proven effective for such correct-by-construction synthesis of controllers for a wide range of applications. This is due in part to the existence of efficient algorithms for the Generalized Reactivity (GR(1)) fragment of LTL, based on finding a winning strategy in a two player game between the controlled system and uncontrolled environment. However, scalability is still a challenge, as these methods scale exponentially in the number of variables in the domain.

Receding horizon control is a common approach to battling the curse of dimensionality in control problems and has been shown to be effective not only in terms of complexity, but also in robustness with respect to exogenous disturbances and modeling uncertainties [12]. The approach involves iterative, short horizon solutions, using the currently observed state to compute a control strategy for some manageable time horizon in the future. Only the first step of the computed strategy is implemented, and new calculations are performed on the next horizon, using the resulting observations.

A receding horizon framework was recently introduced to mitigate the computational intractability of reactive synthesis for temporal logic [18]. The authors propose a reactive synthesis scheme for specifications with GR(1) goals, which relies on partitioning the state space into a sequence of short horizon problems, such that the global problem is realizable if each of the short horizon problems is realizable. Realizability of the short horizon specifications is determined symbolically, but controllers are only extracted as needed, i.e. if and when the respective partitions are reached. A major limitation of this approach is that it relies on user input to provide a priori a pre-determined sequence of short horizon problems for reaching the currently pursued global goal, and does not allow this path to change during execution. This places strong restrictions on the short horizon problems, as described in Section 2, requiring them to have a single point of exit that is reachable in all adversarial environments.

In this paper, we introduce a receding horizon framework that allows the path over short horizon problems to change

automatically in response to the environment. As we illustrate in Section 4, this relieves the user of the burden of defining paths over short horizons, and instead allows them to input just the set of possible next short horizon problems for each short horizon problem. Each short horizon problem now has multiple exits, and the controller can choose one in response to the environment at runtime. Our synthesis algorithm provides this reactive strategy for switching between short horizon problems in a manner that achieves the global goals. As we show in Section 4, another highly advantageous consequence of this approach is that it allows the short horizon problems to be smaller in practice.

In addition to the approach in [18], which we here extend, there have been a few other attempts at using receding horizon control in the context of reactive synthesis from temporal logic specifications. For example, the authors in [7] also propose a receding horizon scheme for specifications in synthetically co-safe LTL. In [5], the authors consider full LTL but use an automata-based approach, involving potentially expensive computations of a finite state abstraction of the system and a Büchi automaton for the specification. We circumvent these expensive operations using symbolic techniques where possible during synthesis. The authors of [5] also restrict their attention to deterministic systems, i.e., those with non-adversarial, deterministic environments, whereas we synthesize controllers for systems that are reactive to a (possibly adversarially) changing changing environment. The authors in [15] also propose a receding horizon solution to the problem of controller synthesis from signal temporal logic specifications: absent once again from that setting is the explicit notion of an adversarial environment, although the receding horizon formulation provides some robustness to environmental uncertainty. Also relevant to this work is that presented in [11], where the authors separate feasibility from controller synthesis, and use metrics on the underlying continuous space to produce short-term strategies that can be chained together to provide globally correct behavior. However, their approach still requires computing the set of winning states for the global specification, whereas we break the feasibility checking problem into short horizon computations.

Contributions: Our contribution is a reactive synthesis algorithm based on receding horizon control that advances the state of the art for specifications in the GR(1) fragment:

- We define short horizon problems with multiple local goals, and choose between local goals at runtime in response to the environment, such that the global goals are satisfied. We claim as a key novelty this automatic reactive switching between short horizon problems to satisfy high-level requirements.
- The reactive strategy for switching between short horizon problems is derived by computing a goal-dependent invariant for the short horizon problems, which provides initial conditions on the environment for which each short horizon problem is winning (i.e. can reach the goal).
- We demonstrate the utility of this additional flexibility in grant-response tasks, via a search-and-rescue example.

2. PRELIMINARIES

We address the problem of designing control software for a plant operating in a potentially adversarial, a priori uncertain environment. The guarantees we seek on the system behavior are of the form, the plant satisfies property φ_s for any valid initial state and any admissible environment; we characterize initial state validity and environment admissibility by specifications φ_{init} and φ_e, respectively.

We assume that the controlled state of the plant evolves according to either a discrete-time, time-invariant dynamics

$$s(t+1) = f(s(t), u(t)), \quad u(t) \in U, \quad \forall T \in \mathbb{N}$$

or a continuous-time, time-invariant dynamics

$$\dot{s}(t) = f(s(t), u(t)), \quad u(t) \in U, \quad \forall T \geq 0$$

where U is the set of admissible control inputs, and $s(t), u(t)$ are the controlled state and control signal at time t.

In order to apply formal synthesis techniques to continuous systems like the above, we require a discrete abstraction of the problem, and a formal language for specifying desired properties. We use Linear Temporal Logic (LTL) as the formal specification language.

2.1 Discrete Abstraction

When designing control software for a physical system such as the one described above, which typically has infinitely many states, a common approach is to construct a finite transition system that serves as a discrete abstraction of the system model. This abstraction must be such that the infinite-state system can *simulate* it, i.e. any discrete plan generated on the abstraction can be implemented on the continuous system. See, e.g., [8, 4, 9, 16, 17, 10] for examples of how such an abstraction can be constructed for various types of dynamical systems.

We assume the availability of such a discrete abstraction of the physical system, and let the system state in this abstraction be characterized by a finite number of boolean variables, $V = S \cup E$; here S and E are disjoint sets that represent, respectively, the set of plant variables that are regulated by the control protocol and the set of environment variables whose values may change arbitrarily throughout an execution. Given V, $\mathcal{V} \subseteq 2^V$ is the finite set of states of the system: a state corresponds to a truth assignment to the variables in V. Similarly, let $\mathcal{S}$ and $\mathcal{E}$ be the sets of states of the plant and environment, respectively.

2.2 Linear Temporal Logic

Syntax: Given a set of atomic propositions AP, boolean operators for negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$), and temporal operators next ($\bigcirc$), always ($\Box$) and eventually ($\Diamond$), an LTL formula is defined by the recursive grammar:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \Diamond\varphi.$$

Semantics: LTL is interpreted over infinite sequences of truth assignments $\sigma : \mathbb{N} \to 2^{AP}$. We say that a truth assignment σ satisfies $\pi \in AP$ at time t (denoted $(\sigma, t) \vDash \pi$) if $\pi \in \sigma(t)$, i.e. σ assigns π to `True` at time t. We say $(\sigma, t) \nvDash \pi$

if π is assigned `False` at time t, i.e. $\pi \notin \sigma(t)$. Note that since we equate states with truth assignments in Section 2.1, we can also write $\nu \vDash \pi$ or $\nu \nvDash \pi$ for $\nu \in \mathcal{V}$.

The semantics of an LTL formula is defined recursively according to the following rules

- $(\sigma, t) \vDash \neg\varphi$ iff $(\sigma, t) \nvDash \varphi$
- $(\sigma, t) \vDash \varphi \wedge \psi$ iff $(\sigma, t) \vDash \varphi$ and $(\sigma, t) \vDash \psi$
- $(\sigma, t) \vDash \varphi \vee \psi$ iff $(\sigma, t) \vDash \neg(\neg\varphi \wedge \neg\psi)$
- $(\sigma, t) \vDash \bigcirc\varphi$ iff $(\sigma, t+1) \vDash \varphi$
- $(\sigma, t) \vDash \Diamond\varphi$ iff $\exists t' \geq t$ s.t $(\sigma, t') \vDash \varphi$
- $(\sigma, t) \vDash \Box\varphi$ iff $(\sigma, t) \vDash \neg\Diamond(\neg\varphi)$

We omit the definition of the until operator, but the reader is referred to [3] for the full syntax and semantics of LTL.

LTL provides an expressive language for specifying properties typically studied in the control and hybrid systems domains, including safety and stability, as well as useful generalizations; see e.g. [18] for a discussion of the types of such properties expressible in LTL.

2.3 Reactive Synthesis

An LTL formula φ over V is *realizable* if there exists a finite state strategy that, for every finite sequence of truth assignments to E, provides an assignment to S such that every resulting infinite sequence of truth assignments to V satisfies φ. It is a well known result that such a strategy exists if and only if there is a deterministic finite state automaton that encodes it [14]. The *synthesis* problem is to find such a finite state automaton when one exists.

Definition 1. A *finite state automaton* is a tuple $A = (\mathcal{V}, V_0, \delta)$ where

- $V_0 \subseteq \mathcal{V}$ is a set of initial states.
- $\delta : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$ is the transition relation.

An automaton is *deterministic* if, for every $v \in \mathcal{V}$ and every $e \in \mathcal{E}$, $|\delta(v, e)| = 1$. Unless mentioned explicitly, all automata considered in this work are deterministic. Let $\delta(v) = \{\delta(v, e) \mid e \in \mathcal{E}\}$ denote the set of possible successor states of state v.

Definition 2. Given an LTL formula φ, deterministic automaton $A_\varphi = (\mathcal{V}, V_0, \delta)$ *realizes* φ if $\forall\sigma = v_0 v_1 v_2 ... \in \mathcal{V}^\omega$ such that $v_0 \in V_0$ and $v_{i+1} \in \delta(v_i)$, $\sigma \vDash \varphi$.

2.4 Generalized Reactivity(1)

Reactive synthesis for a general LTL specification is 2EXPTIME complete [14], but the authors of [2] present a tractable algorithm for the Generalized Reactivity(1) (GR(1)) fragment, which admits specifications of the form

$$\left(\psi_{init} \wedge \Box\psi_e \wedge \bigwedge_{i\in I_f} \Box\Diamond\psi_{f,i}\right) \Longrightarrow \left(\Box\psi_s \wedge \bigwedge_{i\in I_g} \Box\Diamond\psi_{g,i}\right), \tag{1}$$

where

1. ψ_{init}, $\psi_{f,i}$ and $\psi_{g,i}$ are propositional formulas of variables from V;
2. ψ_e is a Boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_e^t$ where ψ_e^t is a propositional formula of variables from E that describes the assumptions on the transitions of environment states; and
3. ψ_s is a Boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_s^t$ where ψ_s^t is a propositional formula of variables from V that describes the constraints on the transitions of controlled states.

We call the left hand side of this expression the *assumptions*, and the right side the *guarantees*.

Problem 1 (Reactive Control Protocol Synthesis)**.** Given a system V and specification φ of the form (1), synthesize a control protocol that generates a sequence of control signals $u[0], u[1], ... \in U^\omega$ to the plant to ensure that starting from any initial condition, φ is satisfied for any sequence of environment states.

3. RECEDING HORIZON SYNTHESIS

The main barrier to applying off-the-shelf reactive synthesis algorithms such as the one in [2] to solve Problem 1 is that it is subject to the curse of dimensionality. In the worst case, the resulting finite state machine contains all possible states of the system – this scales exponentially in the number of system variables. This renders the direct application of reactive synthesis impractical for even moderately-sized problems.

The general framing of the reactive synthesis problem requires planning for all possible environment behaviors. However, we observe in many applications that plans are *local*, in the sense that it is not necessary to plan with respect to environment behaviors that do not affect the current portion of the plan. By incorporating information on the environment obtained at runtime, strategy extraction can be delayed until it is needed. Inspired by the theory of receding horizon control, the authors in [18] present a strategy for reducing the computational complexity by solving a sequence of smaller problems, each with a specific initial condition. Then, at runtime, the automaton is extracted for the currently-observed initial condition, and implemented before switching to the next problem.

A major shortcoming of this approach is the need for the sequence of problems to be pre-determined, and moreover for each of these smaller problems to be realizable, subject to any admissible environment. This effectively restricts the

path to the global goal to a single path through smaller problems, reducing robustness to vagaries of the environment. In this section, we present an approach that enables this path to change in a reactive fashion. This has two consequences:

1. reactive switching between short horizon problems enables these problems themselves to be smaller, since they do not have to account for all possible environment behaviors, and
2. goal-dependent invariants reduce conservativeness per a single global invariant, without loss of soundness.

3.1 Online Selection of Short Horizons

Denote the index set of goals as $I_g = \{1, \ldots, n\}$ for some natural number n, and define a corresponding ordered set $(1, \ldots, n)$, which represents the sequence in which the progress properties $\psi_{g,1}, \ldots, \psi_{g,n}$ will be satisfied.

For each $i \in I_g$, suppose there exists a collection of subsets $\mathcal{C}^i = \{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}$ such that $\mathcal{W}_j^i \subseteq \mathcal{V}$ for all $j \in \{0, \ldots, p\}$[1], and a propositional formula Φ^i over variables in V, such that

(a) $\mathcal{W}_0^i \cup \mathcal{W}_1^i \cup \ldots \cup \mathcal{W}_p^i = \mathcal{V}$,

(b) $\psi_{init} \Rightarrow \Phi^1$ is a tautology, i.e., any state $\nu \in \mathcal{V}$ that satisfies ψ_{init} also satisfies Φ^1,

(c) $\psi_{g,i}$ is satisfied for any $\nu \in \mathcal{W}_0^i$, i.e., once the system reaches any state in $\mathcal{W}_0^i$, it accomplishes the goal corresponding to $\psi_{g,i}$,

(d) $((\nu \in \mathcal{W}_0^i) \ \wedge \ \Phi^i) \Rightarrow \Phi^{(i+1) \mod n}$ is a tautology, and

(e) $\mathcal{P}^i := (\mathcal{C}^i, \preceq_{\psi_{g,i}})$ is a partially ordered set defined such that $\mathcal{W}_0^i \prec_{\psi_{g,i}} \mathcal{W}_j^i, \forall j \neq 0$.

We call Φ^i the *invariant* associated with goal $i \in I_g$. For each $i \in I_g$, define a *short-horizon mapping* $\mathcal{F}^i : \mathcal{C}^i \to 2^{\mathcal{C}^i}$ such that $\mathcal{W}_k^i \prec_{\psi_{g,i}} \mathcal{W}_j^i$ for all $j \neq 0$ and $\mathcal{W}_k^i \in \mathcal{F}^i(\mathcal{W}_j^i)$. Informally, every $\mathcal{W}_k^i \in \mathcal{F}^i(\mathcal{W}_j^i)$ is closer to the goal $\psi_{g,i}$ than $\mathcal{W}_j^i$ for $j > 0$.

Formally, with the above definitions of Φ^i, $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ and $\mathcal{F}^i$, we define a *short-horizon* specification Ψ_j^i associated with $\mathcal{W}_j^i$ for each $i \in I_g$ and $j \in \{0, \ldots, p\}$ as

$$\begin{aligned} \Psi_j^i \ \triangleq \ & \Big((\nu \in \mathcal{W}_j^i) \ \wedge \ \Phi^i \ \wedge \ \Box\psi_e \ \wedge \ \textstyle\bigwedge_{k \in I_f} \Box\Diamond\psi_{f,k}\Big) \\ & \Rightarrow \Big(\Box\psi_s \ \wedge \ \Diamond \textstyle\bigvee_{\mathcal{W}_k^i \in \mathcal{F}^i(\mathcal{W}_j^i)} (\nu \in \mathcal{W}_k^i) \ \wedge \ \Box\Phi^i\Big), \end{aligned} \tag{2}$$

where ν denotes the state of the system and ψ_e, $\psi_{f,k}$ and ψ_s are defined as in (1).

We assume that each Ψ_j^i is realizable. An automaton $\mathcal{A}_j^i$ realizing Ψ_j^i provides a strategy for going from a state $\nu \in \mathcal{W}_j^i$ to a state $\nu' \in \mathcal{W}_k^i$ for some $\mathcal{W}_k^i \in \mathcal{F}^i(\mathcal{W}_j^i)$ while satisfying the safety requirements $\Box\psi_s$ and maintaining the invariant Φ^i associated with goal $i \in I_g$.

Receding Horizon Strategy: For each $i \in I_g$ and $j \in \{0, \ldots, p\}$, construct an automaton $\mathcal{A}_j^i$ realizing Ψ_j^i. Let ν denote the current state of the system. The receding horizon strategy is described in Algorithm 1.

Algorithm 1: Receding horizon strategy

```
1  i := 1;
2  while 1 do
3      I := {ĩ ∈ {1,...,n} | ν ∈ W_0^ĩ};
4      while I = I_g do
5          Make a transition according to automaton A_0^i;
6          I := {ĩ ∈ {1,...,n} | ν ∈ W_0^ĩ};
7      while i ∈ I do
8          i := (i + 1) mod n;
9      Set the index j such that ν ∈ W_j^i;
10     while ν ∉ W_0^i do
11         K := {k ∈ {0,...,p} | ν ∈ W_k^i and W_k^i ≺_{ψ_{g,i}} W_j^i};
12         while K = ∅ do
13             Make a transition according to automaton A_j^i;
14             K := {k ∈ {0,...,p} | ν ∈ W_k^i and W_k^i ≺_{ψ_{g,i}} W_j^i};
15         j := k for some k ∈ K;
```

Algorithm 1 ensures that the goals corresponding to $\psi_{g,1}, \ldots, \psi_{g,n}$ are accomplished in the predefined order. Once the goal corresponding to $\psi_{g,n}$ is reached the process repeats, ensuring that for each $i \in I_g$, a state satisfying $\psi_{g,i}$ is visited infinitely often in the execution. Here i represents the index of the goal that the system is currently trying to reach, and j represents the index of automaton A_j^i that the system is currently executing.

We now explain Algorithm 1 in more detail.

- Line 3 updates I to be the set of indices of goals satisfied by the current state ν. Note that some states may satisfy multiple goals.
- Lines 4–8 consider the case where the system reaches the current goal ($i \in I$). If all the goals are satisfied by the current state ($I = I_g$), we execute automaton A_0^i until the system reaches a state that does not satisfy some goal (Line 4-6). Then, Line 7-8 updates i to the index of the next goal for the system to reach.
- Line 9 updates the index j of automaton A_j^i that the system is currently executing. Note that since for any $i \in I_g$, the union of $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ is the set $\mathcal{V}$ of all the states, given any $\nu \in \mathcal{V}$, there exist $j \in \{0, \ldots, p\}$ such that $\nu \in \mathcal{W}_j^i$.
- In Lines 10–15, the system works through the partial order $(\{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}, \preceq_{\psi_{g,i}})$ associated with the current goal until it reaches the current goal ($\nu \in \mathcal{W}_0^i$). Lines 11–15 are where the system executes the current automaton $\mathcal{A}_j^i$ until it reaches a state $\nu' \in \mathcal{W}_k^i$ for some

[1]For the simplicity of the presentation, we assume that there is a common p for all $i \in I_g$. In general, p depends on i.

$\mathcal{W}_k^i \prec_{\psi_{g,i}} W_j^i$. That is, ν' is a state that is "closer" to the current goal, where the "distance" to the current goal is defined by the partial order $(\{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}, \preceq_{\psi_{g,i}})$. Once $\nu' \in \mathcal{W}_k^i$ is reached, the system starts executing automaton $\mathcal{A}_k^i$. This process is repeated until the current goal is reached.

Theorem 1. *Suppose Ψ_j^i is realizable for each $i \in I_g$, $j \in \{0, \ldots, p\}$. Then the receding horizon strategy ensures that the system is correct with respect to the specification (1), i.e., any execution of the system satisfies equation (1).*

Proof. Consider an arbitrary execution $\sigma = \nu_0\nu_1 \ldots$ of the system that satisfies the assumption part of (1). We want to show that the safety property ψ_s holds throughout the execution and a state satisfying $\psi_{g,\tilde{i}}$ is reached infinitely often for all $\tilde{i} \in I_g$.

Since $\psi_{init} \Rightarrow \Phi^1$ is a tautology, it follows that $\nu_0 \vDash \Phi^1$ and the assumption part of Ψ_0^1 as defined in (2) is satisfied. Line 4–6 are executed only if $\nu_0 \in \mathcal{W}_0^{\tilde{i}}$ for all $\tilde{i} \in I_g$, i.e., ν_0 satisfies all the goals. In this case, $\mathcal{A}_0^1$ is executed. There are two possibilities. First, the while loop in Lines 4–6 never terminates. In this case, we get that $\nu_l \in \mathcal{W}_0^{\tilde{i}}$ for all $l \geq 0$ and $\tilde{i} \in I_g$, i.e., each goal $\psi_{g,\tilde{i}}$ is reached infinitely often. In addition, since the assumption part of Ψ_0^1 is satisfied, $\mathcal{A}_0^1$ ensures that the safety property ψ_s holds throughout the execution. Thus, we only have to consider the case where the while loop in Lines 4–6 terminates. Let ν_l, $l > 0$, be the state of the system at which the loop terminates. We know that $\nu_l \notin \mathcal{W}_0^{\tilde{i}}$ for some $\tilde{i} \in I_g$, i.e., ν_l does not satisfy some goal. Since the assumption part of Ψ_0^1 is satisfied, $\mathcal{A}_0^1$ ensures that $\nu_{\tilde{l}} \vDash \psi_s$ for all $\tilde{l} \in \{0, \ldots, l\}$ and $\nu_l \vDash \Phi^1$.

Next, consider Lines 7–8. Suppose $l \geq 0$ is the index such that ν_l is the state of the system at which the execution enters Line 7. From the previous paragraph, we get that $\nu_l \vDash \Phi^1$. Note that the while loop is executed only if the current state is at the current goal, i.e., $\nu_l \in \mathcal{W}_0^1$. In this case, the index i is updated, according to Lines 7–8, to the index of the next goal to be satisfied. It is easy to check that since $\nu_l \vDash \Phi^1$, and $\Phi^{\tilde{i}}$ is defined such that $((\nu \in \mathcal{W}_0^{\tilde{i}}) \wedge \Phi^{\tilde{i}}) \Rightarrow \Phi^{(\tilde{i}+1) \bmod n}$ is a tautology for all $\tilde{i} \in I_g$, it must be the case that at the termination of the while loop, $\nu_l \vDash \Phi^i$.

Next, consider Lines 10–15. Since $\nu_l \vDash \Phi^i$ and i remains constant in this loop, we can follow the proof in [17] to show that this loop terminates at some state $\nu_{l'}$ such that $\nu_{\tilde{l}} \vDash \psi_s$ for all $\tilde{l} \in \{l, \ldots, l'\}$. In addition, $\nu_{l'} \in \mathcal{W}_0^i$; hence, $\nu_{l'} \vDash \psi_{g,i}$. The proof is based on showing that when each $\mathcal{A}_j^i$ is executed (in Line 13), the assumption part of its specification Ψ_j^i is satisfied. As a result, $\mathcal{A}_j^i$ ensures that ψ_s and Φ^i are satisfied throughout its execution. In addition, the finiteness of the set $\{\mathcal{W}_0, \ldots, \mathcal{W}_p\}$ and its partial order condition ensures that the system makes progress towards and eventually reaches the current goal.

At this point, the next iteration of the most outer loop starts. We can repeat the above proof to show that the system eventually reaches the goal associated with each iteration. In addition, between each pair of successive goals, the safety property ψ_s is always satisfied. Since the goal associated with each iteration is updated following the sequence $(1, \ldots, n)$, we can conclude that each goal $\psi_{g,\tilde{i}}, \tilde{i} \in I_g$ is reached infinitely often. □

Remark 1. It is possible to relax the requirement that a sequence $(1, \ldots, n)$ of goals is pre-defined. For example, we can define a set $FG \subseteq I_g$ of indices of possible first goals (rather than having to start with goal 1 as described earlier). In addition, for each goal $i \in I_g$, we can define a set NG_i of possible next goals (rather than having $(i+1) \mod n$ as the only possible next goal). Condition (b) above then needs to require that $\psi_{init} \Rightarrow \Phi^j$ is a tautology for all $j \in FG$. Furthermore, condition (d) above is modified to ensure that at the point where the current goal is reached, the invariant associated with each possible next goal is satisfied: $((\nu \in \mathcal{W}_0^i) \wedge \Phi^i) \Rightarrow \Phi^j$ is a tautology for all $i \in I_g$ and $j \in NG_i$. At run time, the first goal out of all the possible choices in FG and the next goal out of all the possible choices in NG_i can be picked arbitrarily. A sufficient condition to ensure that all the goals are reached infinitely often is that each of the goals is visited within one cycle (in any arbitrary order).

3.2 Implementation

In order to apply the approach described in Section 3.1, we require as input, for every progress property $\psi_{g,i}$, the collection $\mathcal{C}^i$, partial order $\prec_{\psi_{g,i}}$ and short-horizon mapping $\mathcal{F}^i$. Then we synthesize a collection of automata $\mathcal{A}_j^i$ and use Algorithm 1 to switch between them during execution. We now describe a method of constructing $\mathcal{F}^i$ given a collection $\mathcal{C}^i$, and discuss in detail the continuous execution paradigm, including ramifications of the environment assumptions being violated while executing some $\mathcal{A}_j^i$.

For each goal index $i \in I_g$, we first construct a graph $\mathbb{G}^i = (\mathbb{V}, \mathbb{E})$ whose nodes are the elements of $\mathcal{C}^i$, i.e. $\mathbb{V} = \mathcal{C}^i$. For each $\mathcal{W}_j^i$ and each $\mathcal{W} \subseteq \mathcal{C}^i$, we determine realizability of the specification in (2) with $\mathcal{F}(\mathcal{W}_j^i) = \mathcal{W}$; we can do this in an efficient manner by not considering $\mathcal{W}$ for which we have already considered $\mathcal{W}' \subseteq \mathcal{W}$, since the latter represents a strictly weaker specification. If this specification is realizable, we add to $\mathbb{E}$ the edge $(\mathcal{W}_j^i, \mathcal{W}_k^i)$ for each $\mathcal{W}_k^i \in \mathcal{W}$. With this graph, we define $\mathcal{W}_k^i \prec_{\psi_{g,i}} \mathcal{W}_j^i$ if there is a shorter path to $\mathcal{W}_0^i$ in the graph $\mathbb{G}$ from $\mathcal{W}_j^i$ than from $\mathcal{W}_k^i$. Then we define

$$\mathcal{F}(\mathcal{W}_j^i) = \{\mathcal{W}_k^i \in \mathcal{C}^i \text{ s.t } (\mathcal{W}_j^i, \mathcal{W}_k^i) \in \mathbb{E} \text{ and } \mathcal{W}_k^i \prec_{\psi_{g,i}} \mathcal{W}_j^i\}.$$

If $\mathcal{F}(\mathcal{W}_j^i) = \varnothing$ for some $\mathcal{W}_j^i \in \mathbb{G}$, we recompute the invariant Φ^i. Otherwise, we can then apply the approach in Section 3.1 using this $\mathcal{F}^i$.

It remains to define an execution engine for implementing a transition in automaton $\mathcal{A}_j^i$ in Line 13 (or $\mathcal{A}_0^i$ in Line 5) of Algorithm 1. The continuous execution should *simulate* the discrete transition, as defined formally in, e.g. [1]. Examples of how such a control signal can be computed from the discrete plan can be found in, e.g., [18, 4, 9]. The execution engine maintains the current discrete state $\nu \in \mathcal{V}$ and the next discrete state on the selected transition, $\nu' \in \mathcal{V}$. At each time step, it receives the currently observed (continuous) system state s — note that this state should correspond to the abstract state ν. It determines a control signal that

ensures that the continuous execution of the system according to the dynamics in Section 2 eventually reaches a continuous state corresponding to ν', while remaining in states that correspond to $\{\nu, \nu'\}$.

Since the continuous controller simulates the abstract plan, it follows from Theorem 1 that the continuous execution is guaranteed to preserve correctness of the system.

Note that for each short-horizon problem specified by a formula of the form (2), the corresponding automaton $\mathcal{A}_j^i$ is guaranteed to satisfy the guarantee part if and only if the environment and initial condition respect the assumption part. If one of these assumptions is violated, the specification in (2) is trivially satisfied. However, when we identify that an assumption has been violated, we would like to remedy the solution to work under the new assumptions.

We can remove $\mathcal{W}_j^i$ from the graph $\mathbb{G}$ and check that $\mathcal{F}(\mathcal{W}_k^i) \neq \varnothing$ for all remaining $\mathcal{W}_k^i \in \mathcal{C}^i$. If so, we can still use the synthesized automata $\mathcal{A}_k^i$ for $k \neq j$, as long as the initial condition in the global specification, ψ_{init} does not include states in $\mathcal{W}_j^i$. This is in contrast to the approach in [18], which appeals to a higher-level planner to return a new sequence of short horizons to the current goal when the environment assumptions are violated in one of the short horizon problems. Unlike that approach, we do not have to re-compute the pre-order $\mathcal{F}$ if one of the short horizon problems fails in this manner, since we may have other paths to the goal populated by realizable short horizon problems. This results in fewer calls to the higher-level planner that generates $\mathcal{F}$. Note that if we have already passed the very first state of the global execution, we can still safety remove $\mathcal{W}_j^i$ even if it is part of the initial condition ψ_{init}. However, that if any states in $\mathcal{W}_j^i$ satisfy ψ_{init}, we cannot directly reuse the solution for subsequent executions since we have to be able to start execution from $\mathcal{W}_j^i$; in this case, we need to start afresh with a new partition of the states $\mathcal{C}^i$.

4. EXAMPLE

We demonstrate our framework by applying it to a search-and-rescue scenario.

Example 1. Consider the workspace depicted in Figure 1, where the floor plan is divided into 16 rooms. A subject, who needs to be rescued, can exist in any room. The robot's task is to patrol the rooms for subjects, i.e. to "rescue" any subjects by going to the corresponding room.

Boolean variable $R_{i,j}$ is true if the robot is in the room at the intersection of row i and column j. Similarly, $S_{i,j}$ is true if the subject is in the corresponding room. The specification can now be expressed as follows:

- We want to always eventually rescue every subject: $\square\diamondsuit g_i = \square\diamondsuit(S_{i,j} \Rightarrow R_{i,j})$.
- We assume that the subject, once seen, will not disappear until it is rescued: $\square(S_{i,j} \wedge \neg R_{i,j}) \Rightarrow \bigcirc S_{i,j})$, and will disappear when rescued $\square\left((R_{i,j} \wedge S_{i,j}) \Rightarrow \bigcirc\neg S_{i,j}\right)$.
- We assume that there is only one subject at a time: $\square(S_{i,j} \Rightarrow \neg S_{k,l}) \forall i,j,k,l \in [1,4], (k,l) \neq (i,j)$

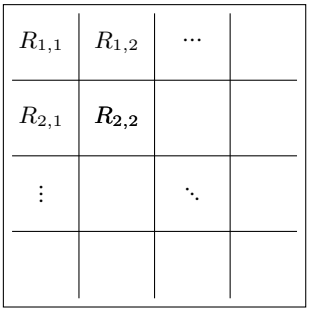

Figure 1: Workspace for Example 1

- A finer discretization of each room, splitting the rooms into several sub-locations and introducing additional dynamics, would make the motivation for a receding horizon approach more apparent. However, this has been omitted here for a simplified presentation, and we assume that the robot (directly) moves from a room to any adjacent room:

$$\square\left(R_{i,j} \Rightarrow \bigcirc N(R_{i,j})\right),$$

where

$$N(R_{i,j}) = \bigvee_{(k,l)\in\{(i,j),(i\pm1,j),(i,j\pm1)\}\cap[1,4]^2} R_{k,l}.$$

- We allow for the possibility that the robot will not be able to transition between two rooms, possibly because of the presence of obstacles in the originating room. We denote the transition between two adjacent rooms being blocked by $B_{(i,j),(k,l)}$:

$$\square\left(B_{(i,j),(k,l)} \Rightarrow \neg((R_{i,j} \wedge \bigcirc R_{k,l}) \vee (R_{k,l} \wedge \bigcirc R_{i,j}))\right),$$

where $R_{k,l} \in N(R_{i,j}), R_{i,j} \neq R_{k,l}$. We allow *one* transition between two adjacent rooms to be blocked:

$$\bigwedge_{\substack{i,j,k,l,i',j',k',l'\in[1,4],\\(i,j,k,l)\neq(i',j',k'l')}} \square(B_{(i,j),(k,l)} \Rightarrow \neg B_{(i',j').(k',l')}),$$

and assume that they will not change when a subject is not rescued

$$\square \quad \left(\bigvee_{i,j\in[1,4]} (S_{i,j} \bigwedge \neg R_{i,j})\right) \Rightarrow \left(\bigwedge_{\substack{i',j',k',l'\in[1,4],\\(k',l')\neq(i',j')}} \left(B_{(i',j'),(k',l')} \leftrightarrow \bigcirc B_{(i',j'),(k',l')}\right)\right).$$

- Finally, we require that a subject is always rescued within a maximum of 6 steps after it appears ($\square(T < 6)$), where the time T is counted according to:

$$\square((\textstyle\bigwedge_{i,j\in[1,4]}(R_{i,j} \vee \neg S_{i,j}) \Rightarrow (\bigcirc T = 0)),$$
$$\square(\textstyle\bigvee_{i,j\in[1,4]}(S_{i,j} \wedge \neg R_{i,j}) \Rightarrow (\bigcirc T = (T+1)).$$

Note that although we have limited our presentation so far to Boolean variable domains, finite integer domains such as that of T are straightforward to implement using a binary encoding, with a number of Boolean variables logarithmic in the size of the domain.

The specifications can be summarized as

$$\varphi_s^e = \bigwedge_{i,j\in[1,4]} \Box(S_{i,j} \bigwedge \neg R_{i,j}) \Rightarrow \bigcirc S_{i,j})$$
$$\bigwedge_{i,j\in[1,4]} \Box(R_{i,j} \bigwedge S_{i,j}) \Rightarrow \bigcirc \neg S_{i,j}$$
$$\bigwedge \Box((\textstyle\bigwedge_{i,j\in[1,4]}(R_{i,j} \vee \neg S_{i,j}) \Rightarrow (\bigcirc T = 0)),$$
$$\bigwedge \Box(\textstyle\bigvee_{i,j\in[1,4]}(S_{i,j} \bigwedge \neg R_{i,j}) \Rightarrow (\bigcirc T = (T+1))),$$
$$\bigwedge_{\substack{i,j,k,l\in[1,4],\\ i',j',k',l'\in[1,4]\\ (i,j)\neq(i',j'),\\ (k,l)\neq(k'l')}} \Box(B_{(i,j),(k,l)} \Rightarrow \neg B_{(i',j'),(k',l')})$$
$$\bigwedge_{\substack{i',j',k',l'\in[1,4],\\ (k',l')\neq(i',j')}} \Box\Big(\big(\bigvee_{i,j\in[1,4]}(S_{i,j} \bigwedge \neg R_{i,j})\big) \Rightarrow \big(B_{(i',j'),(k',l')} \leftrightarrow \bigcirc B_{(i',j'),(k',l')}\big)\Big)$$
$$\bigwedge_{\substack{i,j,k,l\in[1,4],\\ (k,l)\neq(i,j)}} \Box(S_{i,j} \Rightarrow \neg S_{k,l})$$

$$\varphi_s^s = \bigwedge_{i,j\in[1,4]} \Box\,(R_{i,j} \Rightarrow \bigcirc N(R_{i,j}))$$
$$\bigwedge_{i,j\in[1,4]} \Box(B_{(i,j),(k,l)} \Rightarrow \neg((R_{i,j} \bigwedge \bigcirc R_{k,l}) \vee (R_{k,l} \bigwedge \bigcirc R_{i,j})))$$
$$\bigwedge_{\substack{i,j,k,l\in[1,4],\\ (k,l)\neq(i,j)}} \Box(R_{i,j} \Rightarrow \neg R_{k,l})$$
$$\bigwedge \Box(T < 6),$$

$$\varphi_p^s = \bigwedge_{i,j\in[1,4]} \Box\Diamond(S_{i,j} \Rightarrow R_{i,j}),$$

which together with the initial condition

$$\varphi_{init} = (\neg \bigvee_{i,j\in[1,4]} S_{i,j}) \wedge (T = 0),$$

defines the full specification as $\psi = (\varphi_{init} \wedge \varphi_s^e) \Rightarrow (\varphi_s^s \wedge \varphi_p^s)$.

We will now show how our framework can be used to solve this problem efficiently. We focus on the case where we want to fulfil $g_{1,1}$, and the subject is in the corresponding room, i.e. $S_{1,1}$ is true. We define the sets $\mathcal{W} = \{\mathcal{W}_{1,1}, \mathcal{W}_{1,2}, \mathcal{W}_{2,1}...\}$ as:

- $\mathcal{W}_{1,1} = \{\nu \in \mathcal{V} \mid \nu \vDash R_{1,1} \vee \neg S_{1,1}\}$
- $\mathcal{W}_{i,j} = \{\nu \in \mathcal{V} \mid \nu \vDash R_{i,j} \wedge S_{1,1}\}$, for $(i,j) \neq (1,1)$.

We then define the mapping $\mathcal{F}(\mathcal{W})$, as illustrated in Figure 2, as follows, with the mappings for $j > i$ defined symmetrically:

- $\mathcal{F}(\mathcal{W}_{1,1}) = \mathcal{W}_{1,1}$
- $\mathcal{F}(\mathcal{W}_{2,1}) = \mathcal{F}(\mathcal{W}_{2,2}) = \mathcal{W}_{1,1}$
- $\mathcal{F}(\mathcal{W}_{3,1}) = \mathcal{F}(\mathcal{W}_{3,2}) = \{\mathcal{W}_{2,1}, \mathcal{W}_{2,2}\}$
- $\mathcal{F}(\mathcal{W}_{3,3}) = \mathcal{W}_{2,2}$
- $\mathcal{F}(\mathcal{W}_{4,1}) = \mathcal{F}(\mathcal{W}_{4,2}) = \{\mathcal{W}_{3,1}, \mathcal{W}_{3,2}\}$

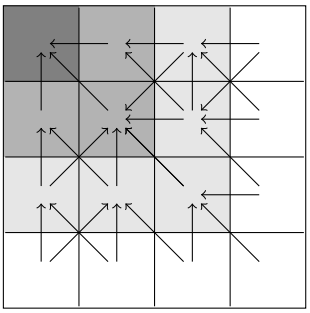

Figure 2: This figure illustrates the mapping $\mathcal{F}$ used in solving the example. Each arrow from $R_{i,j}$ to $R_{k,l}$ represents that $\mathcal{W}_{k,l} \in \mathcal{F}(\mathcal{W}_{i,j})$. The different colors indicate the ordering $\prec_{\psi_{g_{1,1}}}$.

- $\mathcal{F}(\mathcal{W}_{4,3})$=$\{\mathcal{W}_{3,2}, \mathcal{W}_{3,1}\}$
- $\mathcal{F}(\mathcal{W}_{4,4}) = \mathcal{W}_{3,3}$

Finally, we define the ordering as $\mathcal{W}_{i,j} \prec_{\psi_{1,1}} \mathcal{W}_{k,l}$ if and only if $\max(i,j) < \max(k,l)$.

It is now possible to automatically find a sufficient invariant. We start at the goal $\mathcal{W}_{1,1}$ and iterate backwards though the mappings, finding sufficient conditions for reachability for each of the sets $\mathcal{W}_{i,j}$.

- For the last set $\mathcal{W}_{1,1}$, we need $\Phi_{\mathcal{W}_{1,1}} = (T < 6)$ to satisfy all conditions.
- To assure that we can reach $\mathcal{W}_{1,1}$ with $\Phi_{\mathcal{W}_{1,1}}$ from $\mathcal{W}_{2,1}$, we need the additional condition at $\mathcal{W}_{2,1}$ that $\Phi_{\mathcal{W}_{2,1}} = (T < 5 \wedge \neg B_{(2,1),(1,1)}) \vee (T < 3)$.
- For $\mathcal{W}_{2,2}$, we will be able to go to $\mathcal{W}_{1,1}$ with $\Phi_{\mathcal{W}_{1,1}}$, if $\Phi_{\mathcal{W}_{2,2}} = (T < 4)$
- For $\mathcal{W}_{3,1}$, we have two options, either go to $\mathcal{W}_{2,1}$ with $\Phi_{\mathcal{W}_{2,1}}$ or to $\mathcal{W}_{2,1}$ with $\Phi_{\mathcal{W}_{2,2}}$. It is clear that this is possible if $\Phi_{\mathcal{W}_{3,1}} = (T < 2) \vee (T < 4 \wedge \neg B_{(3,1),(2,1)})$.

By continuing this iteration it becomes clear that the invariant

$$\Phi_{g_{1,1}} = \bigwedge_{i,j}((\nu \in \mathcal{W}_{i,j}) \wedge \Phi_{\mathcal{W}_{i,j}}),$$

will guarantee realizability of the short horizon problems as well as $\varphi_{init} \Rightarrow \Phi_{g_{1,1}}$. The same idea can be used for the other goals $g_{i,j}$ to show realizability for the full problem. It should be noted that the robot is allowed to move when setting $T = 0$, which has the consequence that it is possible to reach a target two rooms away at $T = 1$. This sometimes is important for achieving the task within the time bound of 6 in this example. We could restrict motion when resetting the timer if we wanted to be more conservative in this respect.

Figure 3 depicts a solution to the above problem for two environments: one in which the solid red face is blocked and

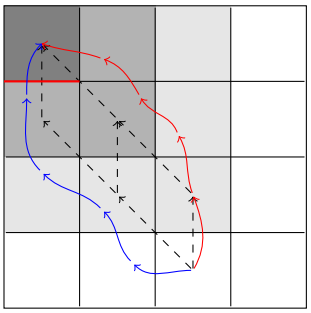

Figure 3: Example of solution with and without blocking. Dashed black lines indicates relevant parts of the mapping. Thick red line indicates that the blue solution is blocked, necessitating the red solution.

one in which is is not: the choice between the two paths is automatic based on the approach presented in Section 3.

A key advantage of using the framework described in Section 3 is that we can keep the sets in $\mathcal{W}$ relatively small compared to the full problem size. The final high level path taken through short horizon problems is decided online, and can therefore depend on the current state of the environment. If we restrict $|\mathcal{F}(\mathcal{W}_i)| = 1$ as in [18], we could be required to group all rooms at a similar distance from the goal in the same set $\mathcal{W}_i$ to allow for different paths to the goal. Doing so enlarges the the short horizon problems and fails to fully exploit the benefits of the receding horizon framework. This effect is magnified when the number of rooms is very large, and when the robot motion planning problem within a room is non-trivial – it would be impractical to consider all rooms at the same distance from the goal in a single problem. Using the approach we have presented, however, we can restrict the search to a few rooms closest to the current position, and choose a subsequent short horizon based on the observed environment.

5. DISCUSSION

We have presented a reactive synthesis framework based on receding horizon control, reducing a synthesis problem over a large domain into a set of significantly smaller problems, and *automatically selects* the next short-horizon problem for the set of all possibilities, based on the observed environment. We presented an example illustrating the approach, and discuss how our method allows the short horizon problems to be smaller in practice than previous attempts at receding horizon control for temporal logic.

This work represents a step towards improving the robustness of the receding horizon framework. Instead of providing a single path to each goal that must work in any environment, the user can provide a set of possibilities, and our synthesis algorithm will automatically determine a feasible path at runtime.

6. ACKNOWLEDGEMENTS

The first author is supported by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

7. REFERENCES

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

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

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

[4] D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas. Valet parking without a valet. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 29 - November 2, 2007, Sheraton Hotel and Marina, San Diego, California, USA*, pages 572–577, 2007.

[5] X. C. Ding, M. Lazar, and C. Belta. LTL receding horizon control for finite deterministic systems. *Automatica*, 50(2):399–408, 2014.

[6] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343 – 352, 2009.

[7] E. A. Gol and M. Lazar. Temporal logic model predictive control for discrete-time systems. In *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*, pages 343–352, 2013.

[8] 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.

[9] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *ICRA*, pages 3116–3121, 2007.

[10] J. Liu and N. Ozay. Abstraction, discretization, and robustness in temporal logic control of dynamical systems. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, HSCC '14, pages 293–302, New York, NY, USA, 2014. ACM.

[11] S. C. Livingston and R. M. Murray. Just-in-time synthesis for reactive motion planning with temporal logic. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 5048–5053, 2013.

[12] R. M. Murray, J. Hauser, A. Jadbabaie, M. B. Milam, N. Petit, W. B. Dunbar, and R. Franz. Online control customization via optimization-based control. In *In Software-Enabled Control: Information Technology for Dynamical Systems*, pages 149–174. Wiley-Interscience, 2002.

[13] P. Nuzzo, H. Xu, N. Ozay, J. Finn, A. Sangiovanni-Vincentelli, R. Murray, A. Donze, and S. Seshia. A contract-based methodology for aircraft electric power system design. *Access, IEEE*, PP(99):1–1, 2013.

[14] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '89, pages 179–190, New York, NY, USA, 1989.

[15] V. Raman, M. Maasoumy, A. Donzé, R. M. Murray, A. Sangiovanni-Vincentelli, and S. Seshia. Model predictive control with signal temporal logic specifications. In *IEEE Conference on Decision and Control (CDC)*, 2014.

[16] P. Tabuada and G. J. Pappas. Linear time logic control of discrete-time linear systems. *IEEE Trans. Automat. Contr.*, 51(12):1862–1877, 2006.

[17] T. Wongpiromsarn, U. Topcu, and R. Murray. Receding horizon temporal logic planning for dynamical systems. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 5997–6004, Dec 2009.

[18] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning. *IEEE Trans. Automat. Contr.*, 57(11):2817–2830, 2012.