Online Horizon Selection in Receding Horizon Temporal Logic Planning

Vasumathi Raman[1] and Mattias Fält[2] and Tichakorn Wongpiromsarn[3] and Richard M. Murray[1]

***Abstract*— Temporal logics have proven effective for correct-by-construction synthesis of controllers for a wide range of robotic 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 goal. We propose a receding horizon algorithm for reactive synthesis that automatically determines a path to the currently pursued goal at runtime, responding as needed to nondeterministic environment behavior. 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, the currently perceived environment and a pre-computed invariant dependent on the 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.**

I. 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], [9], aircraft power systems [14] and smart buildings [16].

Linear Temporal Logic (LTL) has been shown to be an expressive specification language for correct-by-construction controller synthesis for robotics. 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 robotic 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 proven effective not only in terms of complexity, but also in robustness with respect to exogenous disturbances and modeling uncertainties [13]. 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 [20]. 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 all the short horizon problems are 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 II, and requires them to have a single point of exit that is reachable in all adversarial environments.

We introduce a receding horizon framework that allows the path over short horizon problems to change automatically in response to the environment. As illustrated in Section IV, 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 automatically provides this reactive strategy for switching between short horizon problems, such that the global goal is achieved. As we show in Section IV, 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 [20], 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 [8] also propose a receding horizon scheme for specifications in syntactically 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 systems with non-adversarial, deterministic environments, whereas

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

[1]V. Raman and R. M. Murray are with the California Institute of Technology, Pasadena, CA, USA vasu@caltech.edu,murray@cds.caltech.edu

[2]Mattias Fält is with Automatic Control LTH, Lund University, Box 118, SE-221 00 Lund, Sweden faldt.mattias@gmail.com

[3]Tichakorn Wongpiromsarn is with the Thailand Center of Excellence for Life Sciences, Bangkok, Thailand tichakorn@tcels.or.th

we synthesize controllers for systems that are reactive to a (possibly adversarially) changing environment. The authors in [16], [17] propose a receding horizon solution for controller synthesis from a large class of signal temporal logic specifications, for both deterministic and adversarial environments. Their approach is restricted to systems whose dynamics can be encoded as mixed integer linear constraints, and uses mathematical programming to synthesize control inputs. Also relevant to this work is that presented in [12], 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 split the realizability tests 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 in order to satisfy high-level requirements.
- The reactive strategy for switching between short horizon problems is derived by computing a goal-dependent invariant, 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 added flexibility in grant-response tasks, via a search-and-rescue example.

II. Preliminaries

We address the problem of designing control software for a robot operating in a potentially adversarial, a priori uncertain environment: we will guarantee that the robot satisfies its specification for any valid initial state and any admissible environment.

We assume that the controlled state of the robot 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 specification language.

A. 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., [9], [4], [10], [19], [20], [11] 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 robot 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 robot and environment, respectively.

B. Linear Temporal Logic

We use Linear Temporal Logic (LTL) as the formal specification language.

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 ($\Diamond$), LTL syntax is defined recursively as:

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

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 II-A, 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.

When interpreted over a discrete-time finite abstraction of the system, LTL provides an expressive language for specifying properties typically studied in the robotics and hybrid systems control domains, including safety and stability, as well as useful generalizations; see e.g. [20] for a discussion of the types of such properties expressible in LTL.

C. 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. Such a strategy exists if and only if there is a

deterministic finite state automaton that encodes it [15], and 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 $v_{i+1} \in \delta(v_i)$, $\sigma \vDash \varphi$.

D. Generalized Reactivity(1)

Reactive synthesis for a general LTL specification is 2EXPTIME complete [15], but the authors of [2] present a tractable algorithm for the Generalized Reactivity(1) (GR(1)) fragment, which admits specifications of the form

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

where

1) ψ_{init}, $\psi_{f,i}$ and $\psi_{g,i}$ are Boolean formulas over variables in V: $\Box \Diamond \psi_{f,i}$ and $\Box \Diamond \psi_{g,i}$ constitute fairness assumptions on the environment and goal conditions for the system, respectively;
2) ψ_e is a Boolean formula over variables V and expressions of the form $\bigcirc \psi_e^t$ where ψ_e^t is a Boolean formula over variables in E, and describes assumptions on the environment transitions; and
3) ψ_s is a Boolean formula over variables in V and expressions of the form $\bigcirc \psi_s^t$ where ψ_s^t is a Boolean formula over variables in V, and describes constraints on the controlled transitions.

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.

III. 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 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, making the direct application of reactive synthesis impractical for even moderately-sized problems.

The usual 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 new information about the environment at runtime, strategy extraction can be delayed until it is needed. Inspired by receding horizon control, the authors in [20] presented a strategy for reducing computation 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 small problem.

A major shortcoming of this approach is the need for the sequence of small problems to be pre-determined, and moreover for each of these smaller problems to be realizable in any admissible environment. This also 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 each problem can deal with a smaller set of possible environments, and
2) goal-dependent invariants are less conservative than a single global invariant, without loss of soundness.

A. 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 Boolean 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) $((\nu \in \mathcal{W}_0^i) \wedge \Phi^i) \Rightarrow \Phi^{(i+1) \bmod n}$ is a tautology, and
(e) $\mathcal{P}^i := (\mathcal{C}^i, \preceq_{\psi_{g,i}})$ is a partially ordered set defined such that $\mathcal{W}_0^i \prec_{\psi_{g,i}} \mathcal{W}_j^i, \forall j \neq 0$.

For each $i \in I_g$, we 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 $\mathcal{W}_k^i \in \mathcal{F}^i(\mathcal{W}_j^i)$ such that $j \neq 0$. 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

[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.

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

where ν denotes the state of the system and ψ_e, $\psi_{f,k}$ and ψ_s are defined as in (1). We call Φ^i the *invariant* associated with goal $i \in I_g$.

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 $i \in I_g$.

Remark 1. It is possible to further reduce the size of short horizon problems by eliminating locally redundant variables and subformulas, as in [7].

Receding Horizon Strategy: For each $i \in I_g$ and $j \in \{0,\dots,p\}$, 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      Find 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, W_k^i ∈ F^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, W_k^i ∈ F^i(W_j^i)};
15         j := k for some k ∈ K;
```

Algorithm 1 ensures that the goals corresponding to $\psi_{g,1},\dots,\psi_{g,n}$ 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. Since for any $i \in I_g$, the union of $\mathcal{W}_0^i,\dots,\mathcal{W}_p^i$ is the set $\mathcal{V}$ of all the states, given any $\nu \in \mathcal{V}$, there exist $j \in \{0,\dots,p\}$ such that $\nu \in \mathcal{W}_j^i$.
- In Lines 10–15, the system works through the partial order $(\{\mathcal{W}_0^i,\dots,\mathcal{W}_p^i\}, \preceq_{\psi_{g,i}})$ 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 $\mathcal{W}_k^i \in \mathcal{F}^i(\mathcal{W}_j^i)$. Note that $\mathcal{W}_k^i \prec_{\psi_{g,i}} W_j^i$, so ν' is a state that is "closer" to the current goal per the partial order $(\{\mathcal{W}_0^i,\dots,\mathcal{W}_p^i\}, \preceq_{\psi_{g,i}})$. 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 $i \in I_g$, $j \in \{0,\dots,p\}$. 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).*

The proof appears in the technical report available at [18].

Remark 2. It is possible to relax the requirement that a sequence $(1,\dots,n)$ 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 $(i+1) \bmod n$ as the only possible next goal). In place of condition (b) above, we then require that $\psi_{init} \Rightarrow \Phi^j$ is a tautology for all $j \in FG$. Furthermore, condition (d) is modified to ensure that when the current goal is reached, the invariant associated with every possible next goal is satisfied, i.e., $((\nu \in \mathcal{W}_0^i) \wedge \Phi^i) \Rightarrow \Phi^j$ is a tautology for all $i \in I_g$ and $j \in NG_i$. At runtime, 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 using an arbitrary heuristic. A sufficient condition to ensure that all goals are reached infinitely often is that each goal is visited within one cycle (in any arbitrary order).

B. Implementation

In order to apply the approach described in Section III-A, 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 $\mathcal{F}^i$. We then synthesize a collection of automata $\mathcal{A}_j^i$ and use Algorithm 1 to switch between them during execution.

Note that the invariant Φ^i can be computed using the counterexample-driven method described in [20], which is sound but not complete; a complete method remains an open question. 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 $\mathbb{G}^i = (\mathbb{V}, \mathbb{E})$ with $\mathbb{V} = \mathcal{C}^i$. 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 $\mathcal{W}_k^i \in \mathcal{W}$. 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 $\mathbb{G}$ from $\mathcal{W}_j^i$ than from $\mathcal{W}_k^i$. Finally, we set

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

If $\mathcal{F}(\mathcal{W}_j^i) = \varnothing$ for some $\mathcal{W}_j^i \in \mathbb{G}$, we recompute the invariant Φ^i. Otherwise, after processing all goal indices, we can apply the approach in Section III-A using the constructed $\mathcal{F}$.

Remark 3. For many practical applications, it is also possible to automatically construct $\mathcal{C}^i$; e.g., in an autonomous driving scenario, each $\mathcal{W}_j^i$ can be a short road segment.

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 computing such a control signal from the discrete plan can be found in, e.g., [20], [4], [10]. The execution engine maintains the current discrete state $\nu \in \mathcal{V}$ and the next discrete state $\nu' \in \mathcal{V}$ on the selected transition. 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 II eventually reaches a continuous state corresponding to ν', while remaining exclusively 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 can sometimes modify the solution to deal with the new assumptions.

We first 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 [20], which appeals to a higher-level planner to return a new sequence of short horizons problems when the environment assumptions are violated. We do not always have to recompute $\mathcal{F}$ if one of the short horizon problems fails, since we may have other paths to the goal via other 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 safely remove $\mathcal{W}_j^i$ even if it is part of the initial condition ψ_{init}. However, 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 $\mathcal{C}^i$.

IV. Example

We demonstrate our framework using an example motivated by search-and-rescue missions.

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.

Note that the robot can globally sense, e.g., a radio signal indicating a person needing to be rescued. We also make the assumption that when the robot enters the room, it automatically rescues the subject.

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

- Always eventually rescue every subject: $\square \diamondsuit g_{i,j} = \square \diamondsuit (S_{i,j} \Rightarrow R_{i,j})$.
- 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 ($\square\left((R_{i,j} \wedge S_{i,j}) \Rightarrow \bigcirc \neg S_{i,j}\right)$).
- Assume that there is only one subject at a time: $\forall i,j,k,l \in [1,4], (k,l) \neq (i,j),\ \square(S_{i,j} \Rightarrow \neg S_{k,l})$
- 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 $B_{(i,j),(k,l)} \in S$:

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

where $R_{k,l} \in N(R_{i,j}), R_{i,j} \neq R_{k,l}$. We allow *at most 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 the blocked transitions will not change while an already-detected subject has not yet been

rescued:

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

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

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

  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

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

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

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

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 $\psi = (\varphi_{init} \wedge \varphi_s^e) \Rightarrow (\varphi_s^s \wedge \varphi_p^s)$.

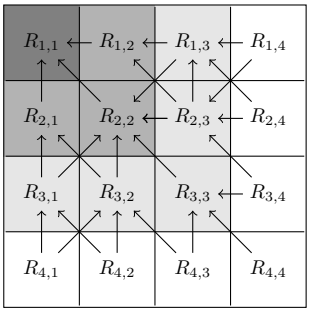

Fig. 1. Illustration of the mapping $\mathcal{F}$ used in solving the example. Each arrow from $R_{i,j}$ to $R_{k,l}$ represents that $\mathcal{W}_{k,l} \in \mathcal{F}(\mathcal{W}_{i,j})$. The different colors indicate the ordering $\prec_{\psi_{g_{1,1}}}$.

We now demonstrate the applicability of our framework on this problem. We focus on the case where we want to fulfill $g_{1,1}$, and the subject is in the corresponding room, i.e. $S_{1,1}$ is true. We define the sets $\mathcal{C}^1 = \{\mathcal{W}_{1,1}, \mathcal{W}_{1,2}, \mathcal{W}_{2,1} ...\}$ 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 $\mathcal{F} : \mathcal{W} \to 2^{\mathcal{W}}$ (illustrated in Figure 1) as follows, with mappings for $j > i$ defined symmetrically:

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

Finally, define $\mathcal{W}_{i,j} \prec_{\psi_{1,1}} \mathcal{W}_{k,l} \Leftrightarrow \max(i,j) < \max(k,l)$.

It is now possible to systematically 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 ensure that we can reach $\mathcal{W}_{1,1}$ with $\Phi_{\mathcal{W}_{1,1}}$ from $\mathcal{W}_{2,1}$, we need an additional condition: $\Phi_{\mathcal{W}_{2,1}} = (T < 5 \wedge \neg B_{(2,1),(1,1)}) \vee (T < 3)$.
- From $\mathcal{W}_{2,2}$, we can reach $\mathcal{W}_{1,1}$ 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 $\Phi_{\mathcal{W}_{2,2}}$. This is achievable if $\Phi_{\mathcal{W}_{3,1}} = (T < 2) \vee (T < 4 \wedge \neg B_{(3,1),(2,1)})$.

By continuing this iteration, we compute the invariant

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

which guarantees 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

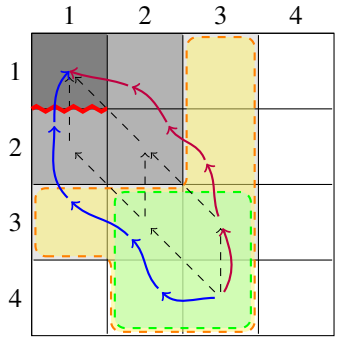

Fig. 2. Example of resulting path with and without blocking. Dashed black lines indicates edges between relevant parts of the mapping. Red zigzag indicates that the blue solution is blocked, necessitating the purple solution. The shaded green area (rows 3-4, cols 2-3), shows a sufficient planning horizon for the starting point in cell $(4,3)$ using the proposed approach, whereas the yellow area represents the original partition without the flexibility afforded by online horizon selection.

in this example. We could restrict motion when resetting the timer if we wanted to be more conservative in this respect.

Figure 2 depicts a path resulting from applying our approach to the above problem, for two different environments: one in which the face between cells $(1,1)$ and $(2,1)$ is blocked, and one in which is is not: the choice between the two paths is automatic.

A key advantage of using the framework in Section III is that we can keep the sets in $\mathcal{C}^1$ relatively small compared to the full problem size (compare the green and yellow shaded areas in Fig. 2). The final high-level path taken through short horizon problems is chosen online, and can therefore depend on the current state of the environment. If we restricted $|\mathcal{F}(\mathcal{W}_{i,j})| = 1$ as in [20], we would be required to group all rooms at a comparable distance from the goal in the same set $\mathcal{W}_{i,j}$ to allow for different paths to the goal. In Figure 2, this corresponds to all rooms with the same shade of grey being part of the same short horizon. 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. Using the approach we have presented, we can divide the set of rooms into smaller subsets, and choose a subsequent short horizon based on the observed environment. Figure 2 demonstrates this advantage: the short-horizon problems are all of size 4 rooms or smaller, where previously the largest problem was of size 7.

V. Discussion

We have presented a reactive synthesis framework based on receding horizon control, reducing the synthesis problem over a large domain into a set of much smaller problems. We significantly improve the robustness of the approach, such that instead of providing a single path to each goal, the user can provide a set of possibilities, and our algorithm will automatically determine a feasible path at runtime. We illustrated the power of our approach with an example, and discussed how our method allows the short horizon problems to be smaller in practice than previous attempts at receding horizon control for temporal logic.

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. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[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, 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] M. Fält, V. Raman, and R. M. Murray. Variable elimination for scalable receding horizon temporal logic. In *American Control Conference, ACC 2015, Chicago, IL, USA, July 1-3, 2015*, 2015.

[8] 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.

[9] 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.

[10] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *2007 IEEE International Conference on Robotics and Automation, 10-14 April 2007, Roma, Italy*, pages 3116–3121, 2007.

[11] J. Liu and N. Ozay. Abstraction, discretization, and robustness in temporal logic control of dynamical systems. In *17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'14, Berlin, Germany, April 15-17, 2014*, pages 293–302, 2014.

[12] 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.

[13] 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.

[14] P. Nuzzo, H. Xu, N. Ozay, J. B. Finn, A. L. Sangiovanni-Vincentelli, R. M. Murray, A. Donzé, and S. A. Seshia. A contract-based methodology for aircraft electric power system design. *IEEE Access*, 2:1–25, 2014.

[15] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190, 1989.

[16] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*, pages 81–87, 2014.

[17] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pages 239–248, 2015.

[18] V. Raman, M. Fält, T. Wongpiromsarn, and R. M. Murray. Online horizon selection in receding horizon temporal logic planning. Technical report, California Institute of Technology, 2015. Full version: `http://resolver.caltech.edu/CaltechCDSTR:2015.001`.

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

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