HOT-SWAPPING ROBOT TASK GOALS IN REACTIVE FORMAL SYNTHESIS

SCOTT C. LIVINGSTON AND RICHARD M. MURRAY

Abstract. We consider the problem of synthesizing robot controllers to realize a task that unpredictably changes with time. Tasks are formally expressed in the GR(1) fragment of temporal logic, in which some of the variables are set by an adversary. The task changes by the addition or removal of goals, which occurs online (i.e., at run-time). We present an algorithm for mending control strategies to realize tasks after the addition of goals, while avoiding global re-synthesis of the strategy. Experiments are presented for a planar surveillance task in which new regions of interest are incrementally added. Run-times are empirically shown to be favorable compared to re-synthesizing from scratch. We also present an algorithm for mending control strategies for the removal of goals. While in this setting the original strategy is still feasible, our algorithm provides a more satisfying solution by "tightening loose ends." Both algorithms are shown to yield so-called reach annotations, and thus the control strategies are easily amenable to other algorithms concerning incremental synthesis, e.g., as in previous work by the authors for navigation in uncertain environments.

1. Introduction

The classical view of formal synthesis is as a two-step process, in which one first specifies a task formally in linear temporal logic (LTL) and then constructs a finite-memory strategy to ensure that the specification is met, despite any external inputs [18] (also cf. the original statement in [5] concerning this two-part view). The resulting strategy usually takes the form of a finite-state machine (or automaton; precise definitions are given in Section 2), which can be deployed with confidence of correctness provided the task does not change and all environmental assumptions remain valid. While this approach is often viable for digital communication protocols, as studied in the computer-aided verification literature, it clearly is not in robotics, where uncertainty and evolving task details are commonplace. This distinction in prerequisites for success between "pure" computer algorithms and deployed robotics systems is demonstrated well by comparing the optimal search algorithms A* and D* [21] (for an introduction to A*, consult e.g., [19]).

Early uses of correct-by-construction control synthesis in robotics thus required strong restrictions against possible sensing or actuation uncertainty [14], [13], [11]. As a precursor to recent work concerning relaxations of these restrictions, the degradation of such "perfect-world" controllers in the presence of sensing uncertainty

S.C. Livingston and R.M. Murray are with the California Institute of Technology, Pasadena, USA. Email address of SCL is slivingston@cds.caltech.edu. This is an extended version of a paper to be presented at the IEEE Conference on Decision and Control (CDC) in Los Angeles, USA, on 15–17 December 2014.
http://resolver.caltech.edu/CaltechCDSTR:2014.001.

is explored in [10]. Examples of relaxations considered in recent work include weakened time synchronization requirements in distributed applications [4], online changes to the workspace cell decomposition [16], and mapping of initially unknown planar workspaces (assuming perfect localization) [20], [23]. In this paper we are concerned with *exact* methods, though we note that there is extensive work concerning Markov decision processes subject to (probabilistic) temporal logic specifications.

Besides uncertainties arising from hardware for sensing and actuation and due to missing details (e.g., maps) about the workspace of the robot, the task itself may be a source of uncertainty. In this paper, a task can be uncertain in the sense that it is not entirely fixed before deployment. Note that this is not merely a problem of resolution at which the task is described: we hope to automatically obtain correct-by-construction controllers and thus we are constrained to framing the task requirements in a way amenable to relevant formal synthesis algorithms. A basic part of most robot architectures is the *supervisor*, a finite-state machine that governs action selection at the level of tasks: "go here", then "if A is present, then release probe f." While there is always a granularity at which the task can be expected not to change during execution, we argue that this granularity may be too coarse to yield useful supervisors. For example, Sarid and collaborators have described an algorithm for realizing a task that is quantified over rooms of certain types (classroom or office) [20]. They consider a mapping application and thus the locations and types of rooms are *a priori* unknown. Their approach is distinct from the first algorithm in the present paper because strategies must be newly entirely created upon discovery of each new entity (room).

Relevant to the present work is [6], in which the authors describe a method for realizing a task expressed as an LTL formula while capturing transient rewards that are discovered online. While in the present paper we focus on online adjustments to the task formula itself, [6] presents a method for ensuring a task (or LTL "specification") is satisfied while allowing for temporary excursions for reward collection.

In this paper, we are concerned with reactive tasks, which can also be viewed as two-player zero-sum games. Here, we mean "reactivity" in the sense used throughout the formal methods literature [18], rather than the sense commonly used in robotics (e.g., as in "reactive motion control", like in [6]). I.e., synthesis results in *strategies* that guarantee correctness when facing any adversarial environment, subject to fairness assumptions. This is distinct from control synthesis that seeks to find a controller such that all executions under that controller meet a specification in the absence of an adversary, as in [6] and elsewhere.

The paper is organized as follows. In Section 2 we outline relevant background material and provide definitions crucial in the present work. This is followed by problem statements concerning online goal additions and removals. Solutions to the problems are presented in separate parts. First, in Section 3, we present and analyze an algorithm for mending strategies for the addition of new goals. Second, in Section 4, we describe what is, informally speaking, the inverse operation: removal of goals online. Finally, simulation experiments are presented in Section 5, followed by a physical validation involving localization, mapping and surveillance in Section 6.

2. Preliminaries and problem formulation

We briefly introduce linear temporal logic (LTL), labeled transition systems and the notion of formal synthesis, mostly to fix notation. For introductions to these topics, the reader is directed to [2] and [9].

Let $\mathcal{X}$ be a set of *environment* variables, and $\mathcal{Y}$ a set of robot (or *system*) variables. Assignments of values to these variables are called *discrete states*, or simply, states. The set of all discrete states is denoted by Γ. We assume that each variable has a finite domain, i.e., in each state it can only be assigned one of finitely many values. If an arbitrary ordering is assigned to the variables $\mathcal{X} \cup \mathcal{Y}$, then Γ becomes the product of the variable domains, and states are ordered tuples. The restriction of Γ to a subset of variables is indicated by a subscript of that set, e.g., the set of environment states is $\Gamma_{\mathcal{X}}$.

LTL builds on propositional (Boolean) logic to describe properties of infinite sequences. The crucial temporal operators for the present work are $\Box$ ("always"), $\Diamond$ ("eventually"), and $\bigcirc$ ("next"). E.g., the boolean formula (or *state formula*) $x = 1$ asserts that the variable x takes the value 1. Notice the lack of assertion about when it will happen. The LTL formula $\Box(x = 1)$ asserts that for all time $x = 1$. $\Diamond(x = 1)$ asserts that a state will eventually be reached where $x = 1$. $\bigcirc(x = 1)$ is true if $x = 1$ at the next time step. Repeatedly reaching states where $x = 1$ is expressed by $\Box\Diamond(x = 1)$.

In this paper, a *task* is an LTL formula over variables in $\mathcal{X}$ and $\mathcal{Y}$. A task is usually provided together with a dynamical systems model of a robot and a workspace that is labeled with some of the system variables. In previous work such labeled cells in the workspace have been referred to as "locative propositions" [15].

We focus on task formulae from the fragment GR(1) [12], [3]. These are of the form

$$(1) \quad \theta_{\text{env}} \wedge \Box\,\rho_{\text{env}} \wedge \left(\bigwedge_{j=0}^{m-1} \Box\Diamond\,\psi_j^{\text{env}}\right) \implies \theta_{\text{sys}} \wedge \Box\,\rho_{\text{sys}} \wedge \left(\bigwedge_{i=0}^{n-1} \Box\Diamond\,\psi_i^{\text{sys}}\right)$$

which is said to be of an assume-guarantee form. θ_{env} and θ_{sys} are initial conditions determining states from which solution trajectories can begin. On the left-side of the implication of (1), ρ_{env} is a formula written in terms of $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X}$, where $\bigcirc\mathcal{X}$ indicates environment variables at the next time step. Intuitively, ρ_{env} constrains how the environment can move given the current state, and as such, it is often called a transition rule. Taken together with the $\Box$ operator applied to it, $\Box\,\rho_{\text{env}}$ is a safety formula that the environment is assumed to satisfy. The right-side is analogously defined but must be guaranteed in the sense that we want to construct a controller realizing it. The system can move based on the current state and the anticipated environment move, since ρ_{sys} is written in terms of $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X} \cup \bigcirc\mathcal{Y}$

Having now shown the form in which tasks are formally expressed, note that throughout the paper, "goals" or "robot goals" refer to formulae ψ_i^{sys} appearing on the right-side of (1). In the present work, we do not consider specifications in which the only feasible solutions drive the environment to a dead-end. (Such cases are irrelevant for the present work since liveness conditions and goals are then immaterial.)

Let φ be a GR(1) formula. A *strategy automaton* for φ is a triple $A = (V, \delta, L)$, where V is a finite set of nodes, $L : V \to \Gamma$ is a state labeling, and $\delta : V \times \Gamma_{\mathcal{X}} \to$

V is a partial function that determines successor nodes in A given inputs from the environment (i.e., states of $\mathcal{X}$ variables). We abbreviate terminology by also referring to A as a "strategy" or "automaton". Note that A may be regarded as a directed graph (V, E), where the edges E are obtained from δ by enumerating over possible environment moves from each node, as provided by the LTL formula φ. With this graph perspective of automaton A, for any node $v \in V$, the successor and predecessor sets of v are defined in the obvious way, and denoted $\mathrm{Succ}(v)$ and $\mathrm{Pre}(v)$, respectively. A *path* is a finite sequence $\langle v_1, v_2, \ldots, v_K \rangle$ of elements from V such that $(v_k, v_{k+1}) \in E$ for $k \in \{1, \ldots, K-1\}$.

Denote the set of nonnegative integers by $\mathbb{Z}_+$. Given φ of the form (1), a state s is said to be a i-system goal if s satisfies ψ_i^{sys}.

Definition 1 (modified from [16])**.** A *reach annotation* on a strategy automaton $A = (V, \delta, L)$ for a GR(1) formula φ is a function $\mathrm{RA} : V \to \{0, \ldots, n-1\} \times \mathbb{Z}_+$ that satisfies the following conditions. Write $\mathrm{RA}(v) = (\mathrm{RA}_1(v), \mathrm{RA}_2(v))$. Given $p < q$, the numbers between p and q are $p+1, \ldots, q-1$, and if $q \leq p$, then the numbers between p and q are $p+1, \ldots, n-1, 0, \ldots, q-1$.

(1) For each $v \in V$, $\mathrm{RA}_2(v) = 0$ if and only if $L(v)$ is a $\mathrm{RA}_1(v)$-system goal.
(2) For each $v \in V$ and $u \in \mathrm{Succ}(v)$, if $\mathrm{RA}_2(v) \neq 0$, then $\mathrm{RA}_1(v) = \mathrm{RA}_1(u)$ and $\mathrm{RA}_2(v) \geq \mathrm{RA}_2(u)$
(3) For any path $\langle v_1, v_2, \ldots, v_K \rangle$ such that $\mathrm{RA}_2(v_1) = \cdots = \mathrm{RA}_2(v_K) > 0$, there exists an environment goal ψ_j^{env} such that for all $k \in \{1, \ldots, K\}$, $L(v_k)$ does not satisfy ψ_j^{env}.
(4) For each $v \in V$ and $u \in \mathrm{Succ}(v)$, if $\mathrm{RA}_2(v) = 0$, then there exists a p such that for all r between $\mathrm{RA}_1(v)$ and p, $L(v)$ is a r-system goal, and $\mathrm{RA}_1(u) = p$.

Reach annotation was introduced in [16]. A summary of key results for the present work is that a reach annotation can be computed during synthesis in constant time (i.e., does not affect asymptotic complexity), it always exists when there is a strategy automaton, and providing one is sufficient to obtain correctness.

Note that the ordering of system goals is arbitrary but fixed. In other words, if a function satisfies Definition 1 after permuting the goal modes, then we can immediately construct a reach annotation from it using the permutation. It can be shown that any function that is reach annotation up to a permutation of the order of goal modes has the same properties as a reach annotation.

It is well-known that initial conditions in φ lead to a set of nodes in the strategy automaton that are not in a strongly-connected component. In other words, these nodes are transiently occupied in all plays, i.e., there is always a finite time horizon after which they can never be returned to, under any play. While one can always ensure that such a prefix of nodes is present, in practice strategy sizes can be reduced by incorporating this prefix so that it is not transient. Throughout this paper, we assume such a compression has not been performed. This assumed form of the given strategy automata is crucial for correctness results proven in Sections 3.3 and 4.3.

Let ζ be a boolean formula. The set of states satisfying it is denoted $\mathrm{Sat}(\zeta)$. (A common alternative is $[\![\zeta]\!]$.) Conversely, let S be a set of discrete states. The boolean formula that is satisfied precisely on S is denoted χ_S. It should be clear that $\mathrm{Sat}(\chi_S) = S$ and $\chi_{\mathrm{Sat}(\zeta)} = \zeta$.

Let A and B be sets of states, and let φ be of the form (1). The *reachability game* from A to B, denoted $\mathrm{Reach}_\varphi(A, B)$, is the reactive LTL formula

$$\chi_A \wedge \Box \rho_{\mathrm{env}} \wedge \left(\bigwedge_{j=0}^{m-1} \Box \diamondsuit \psi_j^{\mathrm{env}} \right) \implies \Box \rho_{\mathrm{sys}} \wedge \diamondsuit \chi_B.$$

Intuitively it provides the same transition rules and liveness (environment) assumptions as φ, amid the task of reaching some state in B from any initial state in A. When realizable, it admits strategies of a similar form to strategy automata together with an abbreviated form of reach annotation [16].

Note that the GR(1) fragment turns out to be quite expressive [17] and is not far from being the most one would hope for in assume-guarantee reactive synthesis problems [7]. Wolff et al. describe another fragment specifically motivated by robotics problems [24], but it is not clear how their approach could make use of binary decision diagrams (BDDs), as we do in our implementation (cf. Section 5).

In this paper we make use of an objective function on discrete states, which we take to measure distance (but see discussion in Section 3.2). This could arise, for instance, from a discrete abstraction on the workspace and continuous robot dynamics [1], [22]. Since in the scope of the present work we do not need anything else from the underlying dynamical system, we do not present how such abstractions can be obtained and instead refer only to "discrete states" throughout the paper. Finding discrete abstractions in general settings is a topic of current research in the hybrid control systems community.

We are now ready to state the problems solved in this paper. Let φ be a GR(1) formula, as in (1), and let $A = (V, \delta, L)$ be a strategy automaton that realizes φ.

Problem 1. *Given a boolean formula ζ, defined over the same variables as φ, find a strategy automaton realizing φ', the extension of φ that includes $\Box \diamondsuit \zeta$, or determine that φ' is unrealizable.*

Problem 2. *Given an index $i \in \{0, 1, \ldots, n-1\}$, find a strategy automaton realizing φ', the formula obtained by deleting $\Box \diamondsuit \psi_i^{\mathrm{sys}}$ from φ, or determine that φ' is unrealizable.*

3. Adding goals

Intuitively, Problem 1 concerns a task in which a goal, i.e., a desirable state that must be visited by the robot infinitely often, is to be added. Note that in practice if one goal can be added, we would expect it possible for more to be requested. Iterating the statement of Problem 1 provides the more general problem of incremental addition of a sequence of new goals $\zeta_1, \zeta_2, \ldots$. Obviously traditional methods are still applicable here: upon receiving a request to add ζ, we can simply discard the entire original automaton A and synthesize for φ' from scratch. However, in some cases we can reduce the time and amount of computation required, as demonstrated empirically in Section 5.

3.1. Overview.

Before presenting the algorithm, we provide a conceptual overview of it. Let φ be a GR(1) formula with n goals (cf. (1)), and let $A = (V, \delta, L)$ be a strategy automaton realizing it. Let ζ be a boolean formula over the same variables as φ. Omitting initial conditions and transition rules, which are unchanged from

φ, the new task formula φ' in terms of environment liveness and robot goals is

$$\bigwedge_{j=0}^{m-1} \square\diamondsuit \psi_j^{\text{env}} \implies \left(\bigwedge_{i=0}^{n-1} \square\diamondsuit \psi_i^{\text{sys}} \wedge \square\diamondsuit \zeta \right). \tag{2}$$

Recall from Section 2 that there is a reach annotation RA for A, and that RA_1 gives the mode for each node. (Recall RA_1 denotes the first number in the pair $\text{RA}(v)$ for automaton nodes $v \in V$.) Roughly speaking, from any $v \in V$, the strategy is seeking to reach a state that satisfies $\psi^{\text{sys}}_{\text{RA}_1(v)}$, i.e., a state satisfying the goal with index $\text{RA}_1(v)$. Being a correct realization, A ensures that such a state will be reached, provided the environment is fair, at which step the mode of the current automaton node is incremented modulo n. Call the set of nodes where the desired goal is reached $G_{\text{RA}_1(v)}$. We can thus broadly view the strategy automaton as moving among node subsets G_i in which task φ goals are reached. The crux of our algorithm is to find which of the existing goals is nearest to the new goal ζ and then to insert a substrategy that visits ζ-states after G_{i^*} where $\psi^{\text{sys}}_{i^*}$ is that nearest goal. The major steps of the algorithm are as follows.

(1) Suppose that we are given a function on pairs of discrete states, which we later call Dist(). Compute this function over ζ-states paired with ψ^{sys}_0-states, then paired with ψ^{sys}_1-states, etc.
(2) For the original robot goals, $\psi^{\text{sys}}_{i^*}$ and $\psi^{\text{sys}}_{i^*+1}$ (index arithmetic is modulo n), find all nodes in the strategy automaton that are meant to reach it, i.e., satisfy that goal and are the consequence of pursuing that goal or any other within the range of goals met at that node. Call these sets G_{i^*} and G_{i^*+1}, respectively.
(3) Solve a reachability game from G_{i^*} to ζ-states.
(4) Let H be the set of ζ-states actually reached in the previous step. Solve a reachability game from H to G_{i^*+1}.
(5) Delete existing paths in A from G_{i^*} to G_{i^*+1}, and append strategies from the two previous reach games in sequence in their place.

A small illustration of our approach in a trivial, deterministic 3×2 gridworld is Figure 1. The setting in Figure 1 is deterministic because there is no adversarial environment; the robot simply moves among cells as on a 4-connected grid. The original task is to visit $(0,1)$ and $(1,0)$ repeatedly. An automaton A realizing it is given on the left-side of Figure 1. The new goal ζ is cell $(0,0)$, the reaching of which requires that we either modify A or create an entirely new one. In this illustration, our algorithm proceeds basically as follows.

(1) The existing goal cells $(1,0)$ (index 0) and $(0,1)$ (index 1) are equidistant from the new goal $(0,0)$. Thus we arbitrarily select the first, i.e., $i^* = 0$.
(2) Clearly $G_0 = \{v_3\}$ and $G_1 = \{v_1\}$.
(3) The reachability game to go from $L(G_0) = \{(1,0)\}$ to the set of ζ-states (i.e., $\{(0,0)\}$) is obviously solved by a two-node strategy that moves one cell up.
(4) A similar two-node strategy solves the reachability game from $(0,0)$ to $L(G_1) = \{(0,1)\}$.
(5) In the original automaton, the old node v_4 would have been visited after G_0. It is now deleted, and replaced by the strategies found in the previous two steps, applied in sequence.

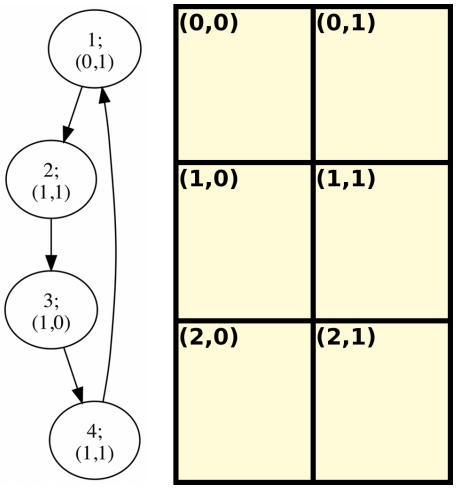

FIGURE 1. Illustrative deterministic example for Algorithm 1. On the right side is a uniform grid discretization of a floor. The original task is to visit cells $(0, 1)$ and $(1, 0)$ infinitely often, and a strategy automaton realizing this is shown on the left. Nodes are referred to by integers. E.g., the topmost node is v_1. The new cell to visit repeatedly is $(0, 0)$.

3.2. **Algorithm.** Our method for online addition of goals is given in Algorithm 1. Details on several parts of it follow.

- line 3: The "distance" function Dist may be more appropriately called an objective function since its purpose is to decide which of the original goals should provide a branching-off point to pursue ζ-states in the substrategies. Thus it can be any heuristic, not necessarily one based on physical distance. When goals correspond to waypoints in a robot workspace, Euclidean distance is a natural heuristic. In our experiments described in Section 5, we use a 1-norm to good effect. Note that our results do not depend on Dist having any particular properties. E.g., we could always select $i^* := 1$.
- lines 4–16: Intuitively, G_{i^*} is the set of nodes where the robot satisfies the task goal $\psi_{i^*}^{\text{sys}}$ and intended to do so. The complicated conditional statement compares changes in the mode, given by RA_1, with i^* to find when this occurs. Recall the definition of reach annotation RA from Section 2.
- lines 21, 27: If one of the reachability games is infeasible, then abort. Note that in general it does not follow that the addition of goal ζ has rendered the task unrealizable. Consult analysis in Section 3.3.
- line 29: Delete original nodes whose use is now replaced by the substrategies found in previous steps. The transition function δ is also updated accordingly.
- lines 29–32: The final steps are to assemble a new strategy automaton A' and a reach annotation RA' for it by mending the original with substrategies $A_{i^*\to\zeta} = (V_{i^*\to\zeta}, \delta_{i^*\to\zeta}, L_{i^*\to\zeta})$ and $A_{\zeta\to i^*+1}$ found by this step in the algorithm. The new labeling $L' : V' \to \Gamma$ is built directly from the

components,

$$L'(v) := \begin{cases} L(v) & \text{if } v \in V, \\ L_{i^* \to \zeta}(v) & \text{if } v \in V_{i^* \to \zeta}, \\ L_{\zeta \to i^*+1}(v) & \text{otherwise,} \end{cases} \tag{3}$$

for $v \in V'$, where it is important to notice that the component node sets are disjoint, i.e., V' is a disjoint union of V, $V_{i^* \to \zeta}$, and $V_{\zeta \to i^*+1}$. RA' is defined in a similar manner,

$$\mathrm{RA}'(v) := \begin{cases} \mathrm{RA}(v) & \text{if } v \in V, \\ \mathrm{RA}_{i^* \to \zeta}(v) & \text{if } v \in V_{i^* \to \zeta}, \\ \mathrm{RA}_{\zeta \to i^*+1}(v) & \text{otherwise.} \end{cases} \tag{4}$$

We omit details concerning the creation of δ'. It is straightforward but tedious (details for a similar process are in [16]).

3.3. **Results.** Here, we prove correctness of Algorithm 1 and describe how the result has a valid reach annotation. We also show that it is not complete and argue why we cannot hope for much better.

Theorem 2. *Let $A = (V, \delta, L)$ be a strategy automaton realizing a GR(1) formula φ, and which has a reach annotation* RA. *Let ζ be a boolean formula over the same variables as φ, and let φ' be the extension of φ to include ζ as a goal. If Algorithm 1 returns a strategy automaton $A' = (V', \delta', L')$ and a map* RA'*, then they are correct with respect to φ', i.e., A' realizes φ' and* RA' *is a reach annotation for A'.*

Proof. The proof proceeds in two steps, first showing that all plays under A' are infinite, and then showing that RA' is a reach annotation, from which correctness follows by Theorem 3 of [16]. Correctness requires infinite plays because, for GR(1) specifications, a finite play is a loss for the robot that occurs when a state is reached from which all possible moves are in violation of the task formula φ' (i.e., would lead into "unsafe" states). (Such states are also known as *dead-ends* in the literature on parity games.)

First, observe that φ' has the same initial conditions and transition rules as φ. We prove by induction on the graph structure of A' that all plays are infinite. For any initial state of φ' (which corresponds exactly to initial states of φ) we can select an initial node v_0 such that $L(v_0)$ is equal to this initial state. Furthermore, by assumption of the form of A described in Section 2, this same node is preserved during the construction of A' from A because it occurs in a prefix of A (i.e., can occur at most once in any play). Since the transition rules are unchanged in φ', the hypothesis that A realizes φ implies that, for any environment move from the state $L(v_0)$, there is an outgoing edge (i.e., a robot move) from v_0 consistent with the transition rules of φ'; otherwise, there would be a play in A that is finite, contradicting the hypothesis of its correctness with respect to φ. For the induction step, let v_k be a node in A' reached under a play $\sigma_0 \cdots \sigma_k$ (so that $L(v_k) = \sigma_k$) that is consistent with the initial conditions and transition rules of φ' (which are the same as those of φ). From Algorithm 1, this node is either a node originating from A, or from one of the substrategies $A_{i^* \to \zeta}$ or $A_{\zeta \to i^*+1}$. In the first case, by hypothesis of A realizing φ, for every possible environment move from σ_k, there is an outgoing edge from v_k consistent with the transition rules of φ', leading to a node v_{k+1} in A'. In the latter case, the definition of reachability games $\mathrm{Reach}_{\varphi'}$ (cf.

Algorithm 1 Append a new goal ζ

INPUT: automaton $A = (V, \delta, L)$, reach annotation RA, distance function Dist, boolean formula ζ
OUTPUT: augmented automaton A' and reach annotation RA$'$
$i^* := \operatorname{argmin}_{i=0,1,\ldots,n-1} \operatorname{Dist}(\psi_i^{\text{sys}}, \zeta)$.
$G_{i^*} := \emptyset$
for all $v \in V$ **do**
  for all $u \in \operatorname{Pre}(v)$ **do**
    if $\big(\operatorname{RA}_1(u) < \operatorname{RA}_1(v) \wedge \operatorname{RA}_1(u) \leq i^* \wedge \operatorname{RA}_1(v) > i^*\big)$
      $\vee\big(\operatorname{RA}_1(u) > \operatorname{RA}_1(v) \wedge (\operatorname{RA}_1(u) \leq i^* \vee \operatorname{RA}_1(v) > i^*)\big)$ **then**
      if $\operatorname{RA}_2(u) = 0$ **then**
        $G_{i^*} := G_{i^*} \cup \{u\}$
      else
        $G_{i^*} := G_{i^*} \cup \{v\}$
      end if
      break //Skip to next iteration of outer for-loop
    end if
  end for
end for
Construct the set G_{i^*+1} in an entirely similar manner to G_{i^*}, but now for the goal mode $i^* + 1$.
if $\operatorname{Reach}_{\varphi'}(G_{i^*}, \operatorname{Sat}(\zeta))$ is realizable **then**
  Synthesize strategy automaton $A_{i^* \to \zeta}$ for the reachability game $\operatorname{Reach}_{\varphi'}(G_{i^*}, \operatorname{Sat}(\zeta))$.
else
  abort
end if
Set G_ζ to all nodes in $A_{i^* \to \zeta}$ that do not have outgoing edges.
if $\operatorname{Reach}_{\varphi'}(G_\zeta, G_{i^*+1})$ is realizable **then**
  Synthesize strategy automaton $A_{\zeta \to i^*+1}$ for the reachability game $\operatorname{Reach}_{\varphi'}(G_\zeta, G_{i^*+1})$.
else
  abort
end if
$V := V \setminus \operatorname{RA}_1^{-1}(i^* + 1)$
$V' := V \cup V_{i^* \to \zeta} \cup V_{\zeta \to i^*+1}$
Set L' and δ' consistent with appending $A_{i^* \to \zeta}$ and $A_{\zeta \to i^*+1}$ to A.
Define RA$'$ on V' so that it agrees with RA on V, $\operatorname{RA}_{i^* \to \zeta}$ on $V_{i^* \to \zeta}$, and $\operatorname{RA}_{\zeta \to i^*+1}$ on $V_{\zeta \to i^*+1}$.

Section 2) ensures that every possible move by the environment under φ' from the state $L(v_k)$, and thus also under φ, has a corresponding outgoing edge in $A_{i^* \to \zeta}$ from node v_k. Mutatis mutandis for $A_{\zeta \to i^*+1}$. Therefore by induction we conclude that all plays of A' are infinite.

Next we show that RA$'$ is a reach annotation for A' with respect to task formula φ'. Let i^* be the index of the goal immediately following which the substrategy reaching ζ-states, $A_{i^* \to \zeta}$, is used. While not done explicitly in Algorithm 1, to

fully conform with the definition of reach annotation (cf. Section 2) we adjust all goal indices (modes) as follows. Set $\psi_n^{\text{sys}} := \zeta$, and let ξ be the permutation of $\{0, 1, \ldots, n-1, n\}$ defined by

$$\xi(\hat{i}) = \begin{cases} \hat{i} & \text{if } \hat{i} \leq i^* \\ n & \text{if } \hat{i} = i^* + 1 \\ \hat{i} - 1 & \text{otherwise.} \end{cases} \tag{5}$$

The robot goals can now be expressed in the usual form, as part of φ,

$$\bigwedge_{\hat{i}=0}^{n} \square \diamondsuit \psi_{\xi(\hat{i})}^{\text{sys}}. \tag{6}$$

Then RA' is a reach annotation using the modes according to (6), i.e., as provided by the permutation (5). Explicitly, define the function RA'' to coincide with RA' on the second part, i.e.,

$$\text{RA}_2''(v) = \text{RA}_2'(v)$$

for all $v \in V'$, and to use permuted modes from RA' on the first part, i.e.,

$$\text{RA}_1''(v) = \xi^{-1}(\text{RA}_1'(v)). \tag{7}$$

Finally, Algorithm 1 appends substrategies for entire modes, i.e., all original nodes with mode $i^* + 1$ are deleted, and the substrategies $A_{i^* \to \zeta}$ are $A_{\zeta \to i^*+1}$ are always followed in sequence, and their nodes have modes n and $i^* + 1$, respectively. Therefore, RA'' is a reach annotation for A' with respect to φ'. Since RA' is the same as RA'' up to a permutation of goal modes, we have that RA' is also a reach annotation for A'. From Theorem 3 of [16], it follows that A' must be winning, i.e., it must realize φ', concluding the proof. □

The new task formula φ' is strictly harder than the original in the sense that any behavior by the robot that meets the new formula necessarily also meets the original, which is intuitively expected given the only change is the addition of a goal to be visited infinitely often. The following remark summarizes this observation.

Remark 3. *Any play that is correct with respect to φ' is correct with respect to φ.*

The previous remark can also be alternatively expressed in terms of language containment, i.e., $\mathcal{L}(\varphi') \subseteq \mathcal{L}(\varphi)$.

Unfortunately Algorithm 1 is not complete, i.e., φ' may be realizable but Algorithm 1 may abort without returning a solution. Intuitively this can occur if the discrete state space can be partitioned into two regions, where in both regions there are strategies for visiting robot goals $\psi_0^{\text{sys}}, \ldots, \psi_{n-1}^{\text{sys}}$, but from one region it is impossible to reach a ζ-state. Then, if A happens to use the wrong region (which is still correct under φ), then no mending of A can be made to realize φ'. One must entirely discard A and instead choose different initial actions to get into the other region. It is not clear how realistic this counter-example sketch is in mobile robot applications.

4. Removing goals

Problem 2 is like an inverse of Problem 1. An initial task has been made simpler by removing one of the goals that was to be repeatedly visited. For example, this could mean that a region of interest in a surveillance task has been permanently

Algorithm 2 Remove an existing goal ψ_i^{sys}

1: INPUT: automaton $A = (V, \delta, L)$, reach annotation RA, deleted goal index i
2: OUTPUT: pruned automaton A' and reach annotation RA$'$
3: Construct the sets G_{i-1}, G_{i+1} in an entirely similar manner as in lines 4–16 of Algorithm 1.
4: Synthesize strategy automaton $A_{i-1\to i+1}$ for the reachability game $\text{Reach}_{\varphi'}(G_{i-1}, G_{i+1})$.
5: $V := V \setminus (\text{RA}_1^{-1}(i) \cup \text{RA}_1^{-1}(i+1))$
6: $V' := V \cup V_{i-1\to i+1}$
7: Define L', δ' consistent with previous line.
8: Define RA$'$ to agree with RA on V and $\text{RA}_{i-1\to i+1}$ on $V_{i-1\to i+1}$.

discarded. While it indeed suffices to continue using A without modification (consult analysis in Section 4.3), this could be wasteful in long- or indefinitely-running robots, where even if the difference in the number of goals added and deleted remains bounded, the strategy automaton itself will grow without bound. Besides, we are practically motivated by keeping strategies succinct when possible. Algorithm 2 solves Problem 2 by pruning the given initial strategy automaton while recovering a reach annotation.

4.1. **Overview.** Before presenting the algorithm and proving properties about it, we provide a conceptual overview. Let φ be a GR(1) formula with n goals (cf. (1)), and let $A = (V, \delta, L)$ be a strategy automaton realizing it. Let $i \in \{0, 1, \ldots, n-1\}$ be the index of the goal removed from the original task formula. Omitting initial conditions, transition rules, and the environment liveness assumptions, which are unchanged from φ, the new task formula φ' in terms of robot goals is

$$\square\diamondsuit \psi_0^{\text{sys}} \wedge \cdots \wedge \square\diamondsuit \psi_{i-1}^{\text{sys}} \wedge \square\diamondsuit \psi_{i+1}^{\text{sys}} \wedge \cdots \wedge \square\diamondsuit \psi_{n-1}^{\text{sys}}. \tag{8}$$

Recall from Section 2 and the discussion in Section 3.1 that the modes provided by the first part of the reach annotation RA_1 indicate the robot goal $\text{RA}_1(v)$ currently being pursued from any automaton node v. This permits viewing the set of nodes as being partitioned according to mode. Thus, if we wish to remove the goal ψ_i^{sys} from the task, we can delete nodes with mode i and mode $i+1$, and replace them with a substrategy that seeks ψ_i^{sys}-states from the loose ends.

4.2. **Algorithm.** Our method for online removal of goals is given in Algorithm 2. Details on several parts of it follow.

- line 4: This reachability game always has a solution, i.e., is realizable. Consult the analysis in Section 4.3.
- line 5–8: The patching process here closely follows that used in Section 3.2, and thus we omit explicit instructions.

4.3. **Results.** Here, we prove correctness and completeness of Algorithm 2 and describe how the result has a valid reach annotation.

As alluded to earlier, after removing a robot goal, the synthesis problem becomes easier in a sense made precise by the following remark. (Compare it with Remark 3.)

Remark 4. *Any play that is correct with respect to φ is correct with respect to φ'.*

Lemma 5. *The reachability game* $\mathrm{Reach}_{\varphi'}(G_{i-1}, G_{i+1})$*, appearing on line 4 of Algorithm 2 is always realizable, i.e., there is at least one substrategy solving it under the transition rules of* φ'

Proof. Recall that the removal of a robot goal from the task does not alter transition rules, nor initial conditions, nor environment liveness assumptions. Therefore, if a state s can be driven by some (finite) strategy to a state t under formula φ, then it can also be done under formula φ' using the same strategy. By hypothesis, the original automaton A realizes φ and in particular reaches G_{i+1} from G_{i-1}, or else blocks an environment liveness condition ψ_j^{env}, and therefore A itself provides a feasible solution to $\mathrm{Reach}_{\varphi'}(G_{i-1}, G_{i+1})$. □

Theorem 6. *Let* A *be a strategy automaton realizing* φ *with reach annotation* RA*. Let* $i \in \{0, 1, \ldots, n-1\}$ *be the index of the goal* ψ_i^{sys} *to be deleted from* φ*, yielding the new task* φ'*. Then* A' *returned by Algorithm 2 on these inputs realizes* φ'*, and* RA' *is a reach annotation.*

Given its similarity to Theorem 2, we only outline the proof. First observe that all plays are infinite, i.e., there are no dead-ends in the automaton A. This is true because the original automaton A has infinite plays from hypothesis, only nodes labeled (via RA) for the deleted goal mode and its successor are removed from A, and the loose ends are connected by a solution of a reachability game that is guaranteed by Lemma 5 to exist. Since all plays are infinite, the second step is to verify that RA' is indeed a reach annotation. This can be shown in a similar manner to that used in the proof of Theorem 2, but now instead of a permutation of modes, we use an injection to account for the gap in the sequence of goal modes due to deletion.

Theorem 7. *Algorithm 2 is complete, i.e., if* φ' *is realizable, then Algorithm 2 will find a strategy automaton* A' *realizing it.*

Proof. It is enough to ensure that Algorithm 2 will terminate with *some* automaton A', because then, by Theorem 6 A' must be correct. This is immediate because Algorithm 2 contains nothing more than for-loops and a reachability game a solution for which always exists by Lemma 5. □

Note that no guarantees are provided about the optimality of strategy automata obtained from Algorithm 2. However, it is easy to ensure that the pruned strategy A' goes from goal with index $i-1$ to that of index $i+1$ in at most as many steps than if we did nothing (i.e., if we kept the original A).

5. Numerical experiment

Implementations of methods described in this paper are provided in gr1c[1], an open-source GR(1) synthesis tool that uses the CU Decision Diagram Package by Fabio Somenzi. A Python interface together with infrastructure for repeating the experiments described here will soon be distributed with TuLiP[2] [25].

5.1. Methods.

Random 4-connected grids—called "gridworlds"—of size 32×32 were randomly generated to contain blocks at a density of 0.2. An initial system

[1] `http://scottman.net/2012/gr1c`

[2] `http://tulip-control.org`

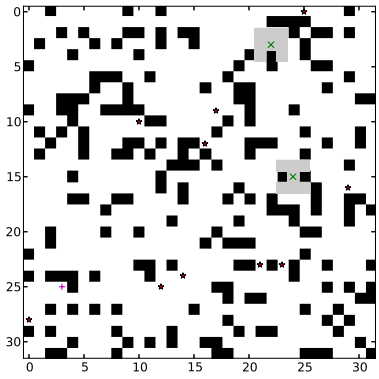

FIGURE 2. Random 32×32 gridworld problem instance. Goal cells to be visited infinitely often are indicated by red stars, the initial position is marked by a magenta plus-sign, and there are two moving obstacles, which are free to move within gray cells and must always eventually return to the cell with a green times-sign (one per gray region).

goal is randomly placed in gridworld, together with 1 or 2 moving obstacles, as illustrated in Figure 2. A nominal control strategy is then obtained. Then, 9 additional system goals are randomly placed in an empty cell. Upon addition of each new goal, Algorithm 1 and global re-synthesis are each applied. Dist() is implemented as the 1-norm.

5.2. **Results.** For the case of one moving obstacle mean times (over 19 trials) of global re-synthesis and patching are shown in Figure 3. For the case of two moving obstacles mean times (over 17 trials) of global re-synthesis and patching are shown in Figure 4.

5.3. **Discussion.** In the case of incremental addition of goals, it is apparent from Figure 3 that the rate of increase in time required to append a substrategy using Algorithm 1 is slower than the rate of increase in global re-synthesis times. This difference suggests that the marginal cost of parsing and manipulating larger strategy automata, as in major steps of Algorithm 1 is much smaller than the marginal cost of constructing and solving an additional goal as part of global re-synthesis. In terms of asymptotic computational complexity, this empirical observation is not theoretically surprising because the two reachability games solved as part of Algorithm 1 (to reach new goal states and then return to the original strategy automaton) have lower alternation numbers than the full GR(1) synthesis problem [3], [8].

The impressive performance gain shown in Figure 3 is somewhat lessened in Figure 4. The only difference between the two settings is the number of moving obstacles. This may be due to the exponentially increasing (global) problem size with the addition of each environment variable, so that for small numbers of system goals, as in this experiment, improvements in speed achieved by our method are dominated by nondeterminism imposed by the adversarial game.

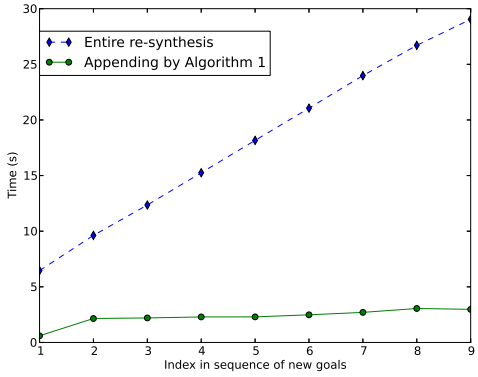

FIGURE 3. Mean run-times for 19 trials of solving randomly generated gridworlds like that depicted in Figure 2. The base task includes one moving obstacle and a single robot goal cell, to be repeatedly visited. Random new goal cells are incrementally added, to reach a final total of 10 robot goals. Upon each goal addition, Algorithm 1 is applied, and re-synthesis (solving from scratch) is also performed. The lower sequence of points are for patching times, whereas the upper sequence is for re-synthesis times.

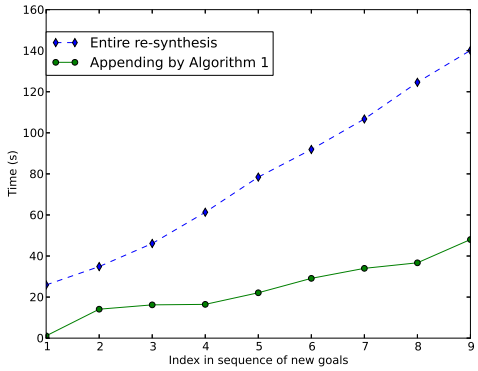

FIGURE 4. Mean run-times for 17 trials of solving randomly generated gridworlds like that depicted in Figure 2, for the case of two moving obstacles. Compare with Figure 3.

6. PHYSICAL VALIDATION

Based on the same implementation used in the simulation experiments described in the previous section, we successfully deployed Algorithm 1 on a differential drive robot equipped with a Hokuyo laser range finder (a modified "TurtleBot"). In summary, we use ROS (www.ros.org) packages providing on-board odometric estimates (wheel encoders and accelerometer of the Kobuki mobile base) together with the ROS gmapping package to proceed in two steps. First, an occupancy grid of the surrounding area is built. After a fixed period of time, we take a snapshot of the map and overlay a coarse gridworld on it, marking cells as blocked if occupancy

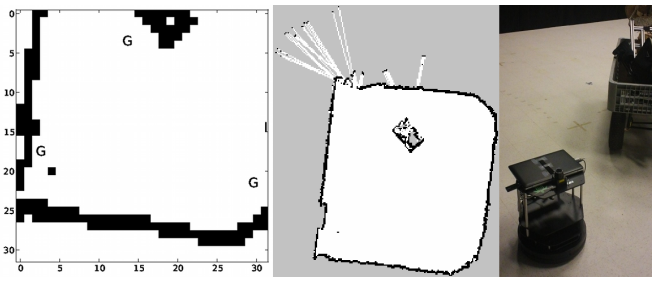

FIGURE 5. Illustration of a run in which the presented method for online goal additions was validated. The right panel shows the robot with mounted range finder and a large static obstacle nearby. The middle panel shows an occupancy grid with cell length of 5 cm. The left panel is a gridworld conservatively built from the lower half of the occupancy grid. Gridworld size is 32×32 with cell length of 20 cm.

probability is above a threshold. The robot then begins surveillance by initializing with three random points of interest (system goals) and visits them infinitely often. New points of interest are randomly added every 30 seconds. A depiction of the setting is shown in Figure 5. The floor space has an area of 6.7 m $\times$ 7.3 m. The initial map-building duration was 7 minutes (420 seconds).

7. Conclusion and future work

In this paper we considered tasks expressed in the GR(1) fragment of LTL, and presented methods to tractably cope with changes to the task due to the addition or removal of robot goals provided incrementally, online. Future work will include long-running physical experiments, based on the preliminary validation described in the present paper. We will also explore construction of interactive systems that use the presented methods to provide for non-adversarial user input by adjusting correct-by-construction automata, which command low-level feedback controllers enforcing discrete abstractions.

As remarked in Section 3.2, none of the results depend on the function Dist. An important topic for future work is studying whether an appropriately chosen Dist can iteratively lead to an optimal solution.

Acknowledgement

This work is partially supported by the Boeing Corporation. The authors thank Vasumathi Raman for motivation.

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, July 2000.

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

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

[4] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, February 2012.

[5] A. Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, Stockholm, August 1962.

[6] X. C. Ding, M. Lazar, and C. Belta. Receding horizon temporal logic control for finite deterministic systems. In *Proceedings of the 2012 American Control Conference*, pages 715–720, Montréal, Canada, June 2012.

[7] R. Ehlers. Generalized rabin(1) synthesis with applications to robust system synthesis. In *Proceedings of NASA Formal Methods Symposium*, 2011. also arXiv:1003.1684v2.

[8] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model checking for the μ-calculus and its fragments. *Theoretical Computer Science*, 258:491–522, 2001.

[9] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[10] B. Johnson, F. Havlak, M. Campbell, and H. Kress-Gazit. Execution and analysis of high-level tasks with dynamic obstacle anticipation. In *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 330–337, Saint Paul, Minnesota, USA, May 2012.

[11] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC)*, pages 2222–2229, Shanghai, China, 2009.

[12] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace inclusion. *Information and Computation*, 200:35–61, 2005.

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

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

[15] H. Kress-Gazit and G. J. Pappas. Automatic synthesis of robot controllers for tasks with locative prepositions. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3215–3220, Anchorage, Alaska, USA, May 2010.

[16] S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray. Patching task-level robot controllers based on a local μ-calculus formula. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4573–4580, Karlsruhe, Germany, May 2013.

[17] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *(PODC '90) Proceedings of the ninth annual ACM Symposium on Principles of Distributed Computing*, pages 377–408, 1990.

[18] 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. ACM.

[19] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, chapter 4, pages 94–134. Prentice-Hall, second edition, 2003.

[20] S. Sarid, B. Xu, and H. Kress-Gazit. Guaranteeing high-level behaviors while exploring partially known maps. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[21] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, August 1995.

[22] P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.

[23] A. Ulusoy, M. Marrazzo, and C. Belta. Receding horizon control in dynamic environments from temporal logic specifications. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[24] E. M. Wolff, U. Topcu, and R. M. Murray. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5018–5025, Karlsruhe, Germany, May 2013.

[25] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. TuLiP: A software toolbox for receding horizon temporal logic planning. In *Proceedings of 14th International Conference on Hybrid Systems: Computation and Control (HSCC'11)*, 2011.