2013 International Conference on Robotics and Automation (ICRA)
http://www.cds.caltech.edu/~murray/papers/lm13-icra.html

Just-in-time synthesis for reactive motion planning with temporal logic

Scott C. Livingston Richard M. Murray

***Abstract*— The cost of the great expressivity of motion planning subject to temporal logic formulae is intractability. Recent advances in sampling-based methods seem to be only applicable to "low-level" control. The problem of realizing "high-level" controllers that satisfy a temporal logic specification does not readily admit approximations, unless the notion of correctness is relaxed as might be achieved with probabilistic variants of temporal logics. In this paper, we argue that not all possible environment (uncontrolled) behaviors need to be explicitly planned for, but rather short-time strategies can be generated online while maintaining global correctness. We achieve this by separating feasibility from controller synthesis, using metrics from the underlying continuous state space to ensure short-time strategies chained together provide globally correct behavior.**

I. Introduction

One major class of approaches to motion planning is that of cell decompositions which ultimately permits reasoning over a discrete abstraction of an otherwise continuous and dynamically constrained problem [1]. In the case of static obstacles, cell decomposition makes path planning a graph search problem while providing guarantees about occupancy of and navigation among cells. Thus, a plan over the graph is executed by steering among these cells.

Beyond moving from an initial configuration to a goal region, we may wish to automatically generate motion plans that cause more sophisticated robot behavior [2]. For instance, synthesize a controller that surveys critical rooms in a building while ensuring battery levels are sufficient. Such "task-level" problems may be expressed in temporal logic formulae. Current research attempting to address this general problem can be roughly grouped according to whether a discrete abstraction is used, and whether the environment is deterministic or uncertain (possibly adversarial). A notable example not using a discrete abstraction is [3]; Karaman and Frazzoli present a sampling-based method to realize a μ-calculus formula in a static environment. Based on earlier results for steering linear systems on polytopic partitions [4], Kloetzer and Belta present a method for realizing linear temporal logic (LTL) formulae defined in terms of the cells in such a partition [5]. Considering robots moving in the plane, Kress-Gazit *et al.* present a method for realizing a class of LTL formulae in the presence of an adversarial environment [6]. A critical component in their work is a synthesis algorithm for computing strategies [7] that solve a game expressed in temporal logic.

Here we are concerned with synthesizing controllers that are correct despite an adversarial environment. The environment is "adversarial" in a robust control sense, i.e., a solution must work against a set of possibilities, and there is uncertainty as to which will actually be encountered. We assume the robot has available to it a discrete abstraction, and that uncontrolled environment actions can be expressed in terms of this abstraction. Viewed as a synthesis problem where all variables have finite domains, realizing such "reactive" LTL formulae has been shown to be intractable [8]. However, an important fragment of LTL known as Generalized Reactivity of rank 1 (GR(1)), can be realized in polynomial time in the number of states [7], [9]. The basic framework is that of a game between the robot system and its environment, with possible moves expressed through safety properties, and robot goals expressed as liveness formulae. A specification is realized by construction of a strategy, which can be represented as a finite automaton.

The present work is not the first to consider an approach motivated by receding horizon control. In [10], a decomposition of states and appropriate augmentation of the specification allow a sequence of short-horizon problems to be solved. However, a major limitation in that work is the need to perform the decomposition manually. So, in some respects, the present work can be regarded as an attempt at formalizing the notion of and automating computation of horizon in [10]. In [11], the authors define a potential function over a product of the transition system and an automaton recognizing a given linear temporal logic formula, such that local "rewards" can be gathered while still making progress down the gradient. An important difference with the present work is that no notion of an adversarial environment appears in [11].

In robotics, a strategy synthesized for a discrete abstraction is implemented by attaching continuous controllers to system "moves" in the game. Thus the robot is controlled as a dynamical system, while input is selected based on a "high-level" view of the state. This notion of implementability is important in relating the continuous and discrete views of the problem; see [12] or [13] for details. Though perhaps not obvious from the above description, synthesizing a *global* strategy is impractical for complex environments and large configuration spaces. A global strategy must account for every possible environment (uncertain, uncontrolled) action, in every possible robot configuration. As in other approaches that rely on an initial discretization, reactive synthesis over a discrete abstraction also suffers from "the curse of dimensionality."

Motivated by modern techniques from symbolic model checking [14], we present a decomposition of the problem of motion planning with GR(1) specifications into

S.C. Livingston and R.M. Murray are with the California Institute of Technology, Pasadena, CA. slivingston@cds.caltech.edu and murray@cds.caltech.edu.

TABLE I
SEVERAL LINEAR TEMPORAL LOGIC (LTL) OPERATORS

$\bigcirc$	"next"
$\diamondsuit$	"eventually"
$\square$	"always" (safety)
$\square\diamondsuit$	"infinitely often" (liveness)
$\models$	"satisfies"

1) feasibility checking, and
2) online short-horizon strategy generation.

Our approach ensures that the chaining together of short-horizon strategies preserves global correctness. Put differently, combining local strategies results in a run through a feasible global strategy. We show that a metric on the robot configuration space can be used to provide sufficient conditions for infinitely often achieving system goals using only local control strategies.

II. PRELIMINARIES

A. Temporal logic and GR(1)

Our notation mostly follows that of [7], with extensions to address robot configuration spaces. We omit some details on temporal logic, and instead refer the reader to [14] for general treatment. Let $\mathcal{V}$ be a set of variables, each of which has some finite domain of possible values, $\operatorname{dom} v$. We assume that $\mathcal{V} = \mathcal{X} \cup \mathcal{Y}$, where $\mathcal{X}$ is a set of environment (adversarial, uncontrolled) variables, and $\mathcal{Y}$ is a set of system (controlled) variables. The system variables $\mathcal{Y}$ are obtained from the configuration of the robot. Thus while being "controlled," the dynamics of the robot constrain how values of these variables may change. For instance, instantaneous teleportation across a map is physically impossible. It will be convenient to refer to the set of all possible states as Γ. Thus

$$\Gamma = \times_{v \in \mathcal{V}} \operatorname{dom} v$$

where $\times$ denotes Cartesian product. Restrictions to possible states of a subset of variables are denoted by a subscript. For example, $\Gamma_{\mathcal{X}}$ is all possible environment states. Given a state $s \in \Gamma$, its projection onto a subset of variables is denoted with a downward arrow, e.g., $s \Downarrow_{\mathcal{X}}$ is the "environment part" of the state s.

The robot and its environment interact in a game. The sequence σ of their moves is called a *play*, written

$$\sigma = s_0 s_1 s_2 \ldots \tag{1}$$

where $s_0, s_1, s_2, \ldots \in \Gamma$ are states occurring at discrete time steps. The rules for evolution of a game and the desired winning conditions are specified in linear temporal logic (LTL). A summary of relevant operators is shown in Table I. We consider here formulae of the type Generalized Reactivity of rank 1 (GR(1)) [7], [9]. A GR(1) specification is of the form

$$\left(\theta_{\text{env}} \wedge \square\, \rho_{\text{env}} \wedge \bigwedge_j \square\diamondsuit\, J_j^{\text{env}}\right) \implies \left(\theta_{\text{sys}} \wedge \square\, \rho_{\text{sys}} \wedge \bigwedge_i \square\diamondsuit\, J_i^{\text{sys}}\right) \tag{2}$$

where θ_{env} and θ_{sys} are propositional formulae describing initial conditions, ρ_{env} and ρ_{sys} are transition rules, and $\mathcal{J}^e = \left\{J_j^{\text{env}}\right\}_{j=1}^{m}$ and $\mathcal{J}^s = \left\{J_i^{\text{sys}}\right\}_{i=1}^{n}$ are goals of the environment and system, respectively.

θ_{env}, θ_{sys}, J_j^{env}, J_i^{sys} are defined over $\mathcal{X} \cup \mathcal{Y}$ and at particular state s are either `True` or `False`. For θ_{env} satisfaction is denoted by $s \models \theta_{\text{env}}$, and similarly for the other formulae. ρ_{env} is defined on $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X}$, and ρ_{sys} on $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X} \cup \bigcirc\mathcal{Y}$. The $\bigcirc$ operator refers to the value of variables at the next time step. Thus given a state s, ρ_{env} specifies the possible environment moves. After an environment (uncontrolled) move is taken, the possible system (robot) responses are provided by ρ_{sys}. Hence, the combined environment and system moves form the next state s', where $(s, s' \Downarrow_{\mathcal{X}}) \models \rho_{\text{env}}$ and $(s, s') \models \rho_{\text{sys}}$

Summarizing the above, an execution $\sigma = s_0 s_1 s_2 \ldots$ is said to satisfy a GR(1) specification φ, denoted $\sigma \models \varphi$, if

1) $s_0 \models \theta_{\text{env}} \wedge \theta_{\text{sys}}$;
2) $(s_i, s_{i+1}) \models \rho_{\text{env}} \wedge \rho_{\text{sys}}$ for $i = 0, 1, \ldots$; and
3) states satisfying J_i^{sys} occur infinitely often, for all system goal conditions in $\mathcal{J}^{\text{sys}}$, or at least one environment goal J_j^{env} is indefinitely blocked.

The left half of expression (2) is an assumption on how the environment will behave. For instance, as a matter of fairness we may assume a street intersection will eventually be clear. Or, we may assume that a charging station will eventually furnish the robot power for its battery. The right half of expression (2) is what must be guaranteed behavior by our controller. For instance, an office robot must infinitely often visit the printer room to service requests. Note that if the left half of expression (2) (assumption) is violated, then desired properties of the synthesized controller are no longer guaranteed.

B. Safety graphs

Using the transition relations ρ_{env} and ρ_{sys} appearing in expression (2), a graph is naturally formed as follows.

Definition 1: Let φ be a GR(1) specification. The graph $G_\varphi = (\Gamma, E)$ with edge set defined by

$$E = \left\{(s, s') \in \Gamma \times \Gamma \mid (s, s') \models \rho_{\text{env}} \wedge \rho_{\text{sys}}\right\},$$

is called the *safety graph* for φ. A state $s \in \Gamma$ for which $s \models \theta_{\text{env}} \wedge \theta_{\text{sys}}$ is called *initial*.

Any execution σ that satisfies a specification φ corresponds to an infinite path through the safety graph G_φ, given that environmental assumptions are met. (For simplicity of presentation, we avoid a game graph formulation [15].)

For completeness, we include some standard definitions when working with such graphs.

Definition 2: Let ψ be a propositional logic formula on variables $\mathcal{V}$. The set of *satisfying states* is

$$\mathrm{Sat}(\psi) = \{s \in \Gamma \mid s \models \psi\}. \quad (3)$$

Definition 3: Let $s \in \Gamma$ be a state in the graph $G_\varphi = (\Gamma, E)$. The set of states reachable in one transition (time step) is

$$\mathrm{Post}(s) = \{s' \in \Gamma \mid (s, s') \in E\}. \quad (4)$$

Let N be a positive integer. The set of states reachable from s in at least 1 transition and at most N is defined recursively

$$\mathrm{Post}^{N+1}(s) = \mathrm{Post}^N \cup \left\{s'' \in \Gamma \mid \exists s' \in \mathrm{Post}^N(s) : (s', s'') \in E\right\}, \quad (5)$$

where $\mathrm{Post}^1(s) := \mathrm{Post}(s)$.

Definition 4: Let $s \in \Gamma$ be a state in the graph $G_\varphi = (\Gamma, E)$. The set of all possible environment actions is

$$\mathrm{EnvPoss}(s) = \{s' \Downarrow_{\mathcal{X}} | \ (s, s' \Downarrow_{\mathcal{X}}) \models \rho_{\mathrm{env}}\}.$$

Note that reachability above depends on unpredictable environment actions. Thus we might wish to restrict the computation of Post^N to a subset of states.

Definition 5: Let $T \subseteq \Gamma$ be a set of states in the graph $G_\varphi = (\Gamma, E)$, and let $s \in T$. The restricted reachable set is defined recursively by

$$\mathrm{Post}_T^{N+1}(s) = \mathrm{Post}_T^N \cup \left\{s'' \in T \mid \exists s' \in \mathrm{Post}_T^N(s) : (s', s'') \in E\right\}, \quad (6)$$

where N is a positive integer, and $\mathrm{Post}_T^1(s) := \mathrm{Post}(s) \cap T$.

C. Strategy synthesis

Given a GR(1) specification φ, the synthesis problem is to find a strategy realizing φ. Beginning from any initial state in G_φ, such a strategy must visit a state for each of the goals in $\mathcal{J}^{\mathrm{sys}}$ infinitely often, despite any permissible environment. One approach [9] to find such a strategy is to

1) compute a fixed point of sets in the graph G_φ for which the system is "winning", i.e., can guarantee satisfaction of all goals infinitely often, and
2) use the intermediate values from the fixed point computation to construct a finite automaton representing a global winning strategy.

For nontrivial problems, synthesizing a global strategy is computationally intensive and may suffer robustness issues in practice. A key observation for the present work is that *realization* (step 2 above) is hard, while *feasibility checking* (step 1) is amenable to symbolic methods, as we now outline.

Consider the goal J_i^{sys}. A fixed point computation on the graph G_φ initializes with

$$T_1^i := \mathrm{Sat}(J_i^{\mathrm{sys}}). \quad (7)$$

Until a fixed point is reached, iterate

$$T_{k+1}^i := T_k^i \cup \big\{s \in \Gamma \mid \forall s' \Downarrow_{\mathcal{X}} \in \mathrm{EnvPoss}(s), \exists s' \Downarrow_{\mathcal{Y}} : ((s, s') \models \rho_{\mathrm{sys}}) \wedge \left(s' \in T_k^i\right) \big\}. \quad (8)$$

Intuitively, each iteration k of equation (8) finds states from which the robot controller can reach a state in T_k^i no matter what the environment does. A similar notion, for instance, is computation of "backwards reachability" despite disturbances. Equations (7) and (8) provide reachability without ensuring that it can be done infinitely often. This and inclusion of environment goals $\mathcal{J}^{\mathrm{env}}$ can be achieved by a μ-calculus formula as shown in [7].

Deciding whether there exists a strategy for a specification of the form (2) is performed by computing the set of winning states T and verifying that all initial states are in T. All steps of the fixed point computation can be performed symbolically, i.e., using ordered binary decision diagrams (OBDDs), which have been demonstrated to concisely represent state sets and transition rules in practice [14]. Roughly, an OBDD of the set T describes the indicator function χ_T, which is defined by

$$\chi_T(s) = \begin{cases} 1 & \text{if } s \in T, \\ 0 & \text{else,} \end{cases}$$

where $s \in \Gamma$. Given an OBDD representation of χ_T and a state s, we can efficiently query whether s is in T.

III. Problem setting and solution method

A. State metrics

Most of the above formalism is standard in model checking and reactive synthesis, as studied in theoretical computer science. However, we are interested in controlling robots that evolve in a state space having more structure than a finite automaton. In particular, a metric is usually available.

Let $\mathcal{V}_{\mathrm{meas}} \subseteq \mathcal{V}$ be the set of variables that have a metric on their domain. Precisely, $v \in \mathcal{V}_{\mathrm{meas}}$ if

$$(\mathrm{dom}\, v, r_v)$$

is a metric space. The metric is $r_v : \mathrm{dom}\, v \times \mathrm{dom}\, v \to \mathbb{R}$. With some abuse of notation, for each $v \in \mathcal{V}_{\mathrm{meas}}$, we extend the domain to include all of Γ as follows. For $s, t \in \Gamma$, project to $s \Downarrow_{\{v\}}, t \Downarrow_{\{v\}} \in \mathrm{dom}\, v$ to compute $r_v(s, t)$. For example, if $y \in \mathcal{Y}$ is the robot configuration, while $\mathcal{V}$ includes system flags that have no metric, then we can use $r_y(s, S_{\mathrm{goal}})$ to measure distance to a region of goal states from the current state by computing the Euclidean distance between $s \Downarrow_{\{y\}}$ and the goal.

Using these component metrics, distances may be computed between any two states $s, t \in \Gamma$ by

$$r(s, t) = \sum_{v \in \mathcal{V}_{\mathrm{meas}}} r_v(s, t), \quad (9)$$

We require that every state not satisfying J_i^{sys} has nonzero distance to $\mathrm{Sat}(J_i^{\mathrm{sys}})$. If this is not the case, the distance function can be extended by using the discrete metric on some or all of the remaining variables $v \in \mathcal{V} \setminus \mathcal{V}_{\mathrm{meas}}$. Explicitly,

$$r_v(s, t) = \begin{cases} 0 & \text{if } s = t, \\ 1 & \text{else.} \end{cases}$$

We can now define the distance to goal states and thus obtain a potential-like function for steering. Let φ be a GR(1) specification (cf. expression (2)), and let $J_i^{\rm sys} \in \mathcal{J}^{\rm sys}$ be a goal condition for the robot to meet infinitely often. Recall that $\mathrm{Sat}(J_i^{\rm sys})$ is the set of states in the safety graph G_φ at which $J_i^{\rm sys}$ is satisfied.

Definition 6: Let $s \in \Gamma$, and let r be as in equation (9). The *distance to goal* $J_i^{\rm sys}$ is

$$\mathrm{distg}_i(s) = \min_{t \in \mathrm{Sat}(J_i^{\rm sys})} r(s,t).$$

B. A sufficient horizon

For clarity, our treatment below focuses on reaching a single robot (system) goal $J_1^{\rm sys}$. Extension to the general case is sketched at the end of this section. We assume the winning set T has already been computed by some method (such as the fixed point computation of [7]) the result of which is its indicator function χ_T. For example, as outlined in the previous section, χ_T may be represented by an ordered binary decision diagram resulting from a symbolic fixed point iteration.

Since the distance function r on states can be computed from the current state, the goal distance distg (from Definition 6) provides the basis for a sufficient condition on horizon length in generating local strategies. Before this can be presented, we define two quantities to be computed at each fixed point iteration (equation (8)).

Remark 1: Let k be a positive integer. Define $T_0 := \emptyset$. Then the set of new points added to the winning set T at step k is $T_k \setminus T_{k-1}$. The shortest guaranteed path length from any state in $T_k \setminus T_{k-1}$ to the goal set is $k-1$.

Remark 2: From finiteness of the safety graph G_φ and monotonicity of the fixed point iteration in equation (8), it follows that the fixed point will be achieved in finitely many steps. Call the index at which the fixed point is achieved k_*. So, $T_{k_*+1} \setminus T_{k_*} = \emptyset$.

Remark 3: Let $s \in T$. Then there is a unique $k \in \{1, \ldots, k_*\}$ such that $s \in T_k \setminus T_{k-1}$

For each $k \in \{1, \ldots, k_*\}$, the minimum and maximum values taken by distg on $T_k \setminus T_{k-1}$ are denoted Min_k and Max_k, respectively. Precisely,

$$\begin{aligned} \mathrm{Min}_k &:= \min_{s \in T_k \setminus T_{k-1}} \mathrm{distg}(s) \\ \mathrm{Max}_k &:= \max_{s \in T_k \setminus T_{k-1}} \mathrm{distg}(s). \end{aligned}$$

Note that these quantities can be found iteratively as the fixed point iterations are computed.

Suppose that the fixed point is achieved at iteration $k_* \geq 2$. (Otherwise, the problem is trivial because the winning set is just the goal, i.e., the robot can only correctly occupy goal states.)

Theorem 1: Let N be a positive integer such that for each $k \in \{2, \ldots, k_*\}$, there exists l where $k - l \leq N$ and for all $l' \leq l$

$$\mathrm{Max}_{l'} < \mathrm{Min}_k \,. \tag{10}$$

Algorithm 1 Compute sufficient horizon N

Require: specification of the form $\diamondsuit J_1^{\rm sys}$.

```
N := 1
for k = 3, ..., k_* do
    for l = k − 2, ..., 1 do
        if Max_l ≥ Min_k then
            N_k := k − l
        end if
    end for
    if N_k > N then
        N := N_k
    end if
end for
```

Let $s \in T_k \setminus T_{k-1}$, and let

$$t^* := \mathrm{argmin}_{t \in \mathrm{Post}_T^N(s)} \mathrm{distg}(t). \tag{11}$$

Then $t^* \in T_q \setminus T_{q-1}$ for some $q < k$.

Proof: If $\mathrm{distg}(t^*) = 0$, then by definition $t^* \models J_1^{\rm sys}$, thus $q = 1$ and we are done. Thus take $\mathrm{distg}(t^*) > 0$. By construction of N (see equation (10)), there exists $l_s \in \{k-N, k-N+1, \ldots, k-1\}$ such that $\mathrm{Max}_{l'_s} < \mathrm{Min}_k$ for $l'_s \leq l_s$. Observe that $k \neq q$ because, from the fixed point iteration defining the sets $T_1, T_2, \ldots, T_{k_*}$, $\mathrm{Post}_T^N(s) \cap T_{l_s}$ is nonempty, hence $\mathrm{distg}(\hat{t}) < \mathrm{distg}(\hat{s})$ for $\hat{s} \in T_k$, $\hat{t} \in \mathrm{Post}_T^N(s) \cap T_{l_s}$.

Suppose to the contrary that $k < q$. Again by construction of N, there exists $l_{t^*} \in \{q-N, q-N+1, \ldots, q-1\}$ such that $\mathrm{Max}_{l'_{t^*}} < \mathrm{Min}_q$ for $l'_{t^*} \leq l_{t^*}$. Now consider the two possible cases.

Case 1. $l_s \leq l_{t^*}$. But then it follows that $\mathrm{Max}_{l_s} < \mathrm{Min}_q$. As observed earlier, there exists $\hat{t} \in \mathrm{Post}_T^N(s) \cap T_{l_s}$, hence $\mathrm{distg}(\hat{t}) < \mathrm{distg}(t^*)$, whence a contradiction (since t^* was the supposed minimum).

Case 2. $l_s > l_{t^*}$. Combining the various inequalities concerning indices above, we have

$$k - N < q - N \leq l_{t^*} < l_s \leq k - 1 < q - 1.$$

In particular, $k - N < l_{t^*} < k - 1$. Similarly to as observed in Case 1, there exists $\hat{t^*} \in \mathrm{Post}_T^N(s) \cap T_{l_{t^*}}$, so it follows that $\mathrm{distg}(\hat{t^*}) < \mathrm{distg}(t^*)$, whence a contradiction. ∎

Remark 4: The horizon N in Theorem 1 is bounded because the winning set is bounded, hence for sufficiently large N, $\mathrm{Post}_T^N(s) \cap \mathrm{Sat}(J_j^{\rm sys}) \neq \emptyset$ for any $s \in T$.

C. Algorithms

Theorem 1 provides that a horizon of length N, where N satisfies the various hypotheses of the theorem, suffices to always find a state that is strictly closer to the goal set. Algorithm 1 provides a method for computing N that satisfies these hypotheses. Informally speaking, $N > 1$ allows for finite move sequences *away* from goal states (so that states reached during the sequence are in $T_k \setminus T_{k-1}$ sets that do not satisfy equation (10)), as may be required, say, to avoid collision with a dynamic obstacle. Combined with a winning set indicator χ_T, we have the basis for short-horizon

online synthesis algorithms. Based on these ingredients, in Algorithm 2 we propose an elementary method, dubbed "Single-step short-horizon."

The basic operation of Algorithm 2 is as follows. During each time step, first observe the action of the environment. For instance, this could be a movement by a dynamic obstacle, or the signalling of a "low battery" flag. The previous state and the present environment action imply, in a way easily computable from the given GR(1) specification, a collection of possible robot actions. From here, the set of states that might be reached within a horizon of N may be computed. The state with minimum goal distance is found over this reachable set, restricted to be a new minimum-distance state if the current state has been visited before. Intuitively, the motivation for memory-based restriction is to prevent environment-forced cycles in which the robot never reaches its goals. The robot action constituting the first step on the shortest path is then taken, and the process repeats.

Note that Algorithm 2 contains a subroutine call, $\mathrm{ShortPath}$, for computing the shortest path between states in a graph. $\mathrm{ShortPath}$ takes three arguments because the current environment action $s'_{\mathcal{X}}$ restricts the set of possible next states for any path from s to t^*.

Theorem 2: Let φ be a specification of the form $\diamondsuit J_1^{\mathrm{sys}}$. Controlling the robot under Algorithm 2 realizes φ. That is, any execution under Algorithm 2 is globally correct with respect to φ.

Proof: It suffices to show that a given state will lead to a state that is strictly closer to the goal set in finitely many steps, since then by induction we obtain the result, i.e., a state satisfying the goal conditions is guaranteed to be reached in finitely many steps.

Let $s \in T$, and let $s' \in T$ be the next state reached after one loop of Algorithm 2. First observe that for all time steps, the robot (system) remains in the winning set T because, by its very construction, no matter what action the environment takes, there will be a possible robot action from which the next state s' is indeed in T. Let k and q be the indices such that $s \in T_k \setminus T_{k-1}$ and $s' \in T_q \setminus T_{q-1}$, respectively.

We wish to show that the case of $k \leq q$ can only occur finitely many times. Suppose to the contrary. Because of the finite memory MEM for visited states, each time a state is repeated, a new target state t^* is selected. However, the set $\mathrm{Post}_T^N(s)$ is finite, and thus removing an additional element from it each time s is visited will in finitely many steps cause all possible robot actions to be tried from s. But by construction of the winning set T, at least one of these actions must lead to a state strictly closer to the goal, whence a contradiction. ∎

The treatment above can be extended to the general case, including multiple system and environment goals as follows.

1) Use our method proposed for each system goal, where the sets $T_1, T_2, \ldots$ are now intermediate sets from the computation of μY in equation (2) of [7] (also see [9]).
2) Take horizon N to be the maximum among all system goal horizons.
3) During online execution, alternate pursuit of each system goal. I.e., first seek a J_1^{sys}-state; once one is reached, then seek a J_2^{sys}-state, etc.

Algorithm 2 Single-step short-horizon strategy

Require: specification of the form $\diamondsuit J_1^{\mathrm{sys}}$.
1: $s \leftarrow$ any initial state
2: $\mathrm{MEM} \leftarrow \emptyset$ //finite memory, for tracking visited states
3: **repeat**
4: Read $s'_{\mathcal{X}}$ //sense environment action
5: **if** $s \in \mathrm{MEM}.keys$ **then**
6: $R \leftarrow \mathrm{MEM}[s]$
7: **else**
8: $R \leftarrow \emptyset$
9: $\mathrm{MEM}[s] \leftarrow \emptyset$
10: **end if**
11: $t^* \leftarrow \mathrm{argmin}_{t \in \mathrm{Post}_T^N(s) \setminus R} \mathrm{distg}(t)$
12: $\mathrm{MEM}[s] \leftarrow \mathrm{MEM}[s] \cup \{t^*\}$
13: $s \leftarrow \mathrm{ShortPath}(s, s'_{\mathcal{X}}, t^*)[1]$ //take first step on shortest path to t^*
14: **until** $s \in \mathrm{Sat}(J_1^{\mathrm{sys}})$

Algorithm 2 can be extended for use with full GR(1) specifications by skipping line 12 if one of the environment goals is not satisfied. This ensures that from a particular state, the environment cannot eliminate all options (i.e., cause $\mathrm{Post}_T^N(s) \setminus \mathrm{MEM}[s]$ to become empty) by not being fair.

IV. A SIMULATION EXAMPLE

We illustrate the proposed method by an example of driving to goal positions in "gridworlds" with dynamic obstacles. A "gridworld" is just a rectangular subset of $\mathbb{R}^2$ that has a uniform cell decomposition, where we assume the robot dynamics allow steering between adjacent cells (thus forming a 4-connected grid). Simulation experiments described in this section are implemented as part of an extension of the GR(1) strategy synthesis tool "gr1c".[1] Binary decision diagrams for symbolic model checking are provided by the CU Decision Diagram package by Fabio Somenzi and contributors.[2]

An intuitive motivation for receding-horizon-like methods is that robot behavior in the next few time steps should be unaffected by things that are sufficiently "far away." In the present work, we studied this in the case of a nondeterministic adversary who is described by linear temporal logic. To see how the horizon computed by our method allows such separation, consider the setting shown in Figure 1. It is a 4-connected grid of size 32×8 with goal positions at opposite corners (marked by **G**). The robot is to infinitely often visit both **G** cells—as in a simple surveillance task—while avoiding collisions with two moving obstacles. The first obstacle can move by at most one cell per time step, though it is restricted to the gray region of rows 1 and 2. The second obstacle is entirely similar to and independent from the first; it is restricted to rows 14 and 15.

[1] `http://scottman.net/2012/gr1c`
[2] `http://vlsi.colorado.edu/~fabio/CUDD/`

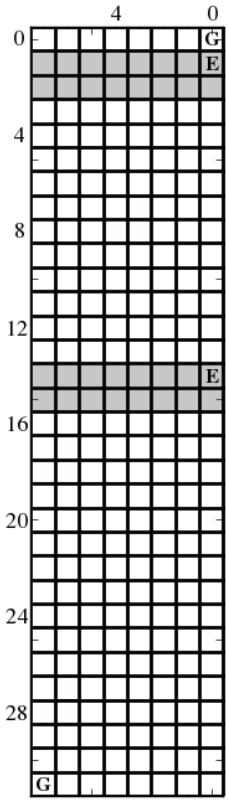

Fig. 1. Layout of the example. The possible positions of two dynamic obstacles are indicated by the two regions of gray cells, i.e., rows 1 and 2; and rows 14 and 15. The goal positions for the robot are indicated by the letter **G**. As a fairness condition, the obstacle in each gray region must infinitely often return to the cell labelled with **E**.

From Algorithm 1 and using the 1-norm to compute distance between the robot position and the goal, the horizon N is computed to be 10. This same horizon is found if either obstacle is excluded. In other words, Algorithm 1 provides the same sufficient planning horizon to guarantee collision avoidance whether we have one or two obstacles in the setting of Figure 1. Given the separation of the gray regions, one may expect this result.

V. Conclusion and future work

We have presented a sufficient condition for selecting the horizon in motion planning with specifications from the GR(1) fragment of linear temporal logic. This permits dividing the problem into two parts: winning set computation, and (online) short-horizon strategy synthesis. This separation is crucial because often χ_T, the winning set indicator function, is efficiently executed and symbolically represented, e.g., using ordered binary decisions diagrams. We also provided an algorithm for on-the-fly strategy generation based on this horizon.

Though our example is simple enough to achieve good performance with the 1-norm, generally the distance function on states may be more sophisticated, e.g., based on navigation functions from [16], which wrap around static obstacles and may lead to small sufficient horizons in cluttered spaces. Experiments with the proposed algorithm using such metrics and on robot hardware will be addressed in future work.

Another topic of future work is to study how a sufficient horizon may be described in terms of local graph structure from the GR(1) specification. Put differently, the discrete abstraction of a task-level planning problem provides its own notion of distance: (possibly weighted) path length in terms of number of edges in a game graph. In the present work, we studied how a metric on states, e.g., from the underlying continuous dynamical system, can inform online strategy generation. Future work will investigate the interaction of these continuous and discrete views of distance to goals.

Acknowledgments

This work is partially supported by the Boeing Corporation.

References

[1] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu/

[2] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion: Finding the missing pieces of current methods and ideas," *IEEE Robotics & Automation Magazine*, pp. 61–70, March 2007.

[3] 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)*, Shanghai, China, 2009, pp. 2222–2229.

[4] L. Habets and J. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, pp. 21–35, 2004.

[5] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. on Automatic Control*, vol. 53, no. 1, pp. 287–297, February 2008.

[6] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

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

[8] 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*, ser. POPL '89. New York, NY, USA: ACM, 1989, pp. 179–190.

[9] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *In Proc. 7th International Conference on Verification, Model Checking and Abstract Interpretation*, ser. Lecture Notes in Computer Science, vol. 3855. Springer, 2006, pp. 364–380. [Online]. Available: http://jtlv.sourceforge.net/

[10] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proceedings of 13th International Conference on Hybrid Systems: Computation and Control (HSCC'10)*, 2010.

[11] X. C. Ding, C. Belta, and C. G. Cassandras, "Receding horizon surveillance with temporal logic specifications," in *Proceedings of the 49th IEEE Conference on Decision and Control (CDC)*, December 2010, pp. 256–261.

[12] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, July 2000.

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

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

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

[16] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, October 1992.