Submitted, 2014 Internatioanl Conference on Robotics and Automation (ICRA)
http://www.cds.caltech.edu/~murray/papers/lm14-icra.html

Moving bounding boxes and incremental synthesis for dynamic obstacles

Scott C. Livingston Richard M. Murray

***Abstract*— While the use of formal synthesis for robotics problems in which the environment may act adversarially provides for exact—rather than probabilistic—correctness of controllers, such methods are impractical when the adversary can move freely in a large portion of the workspace. As is well-known, this is due to exponential growth in the state space with the addition of each new problem variable. Furthermore, such an approach is overly conservative because most configurations will not be reached in typical runs. Rather than entirely abandon the discrete game view, we propose a combined method that ensures exact satisfaction of a given specification, expressed in linear temporal logic, while providing a lower bound on robot-obstacle distance throughout execution. Our method avoids explicit encoding of the moving obstacle and thus substantially reduces the reactive synthesis problem size, while allowing other nondeterministic variables to still be included in the specification. Our approaches centers on modeling obstacle motion as changes in the presence of a virtual static obstacle, and performing incremental synthesis in response. The algorithm is tested in application to a planar surveillance task.**

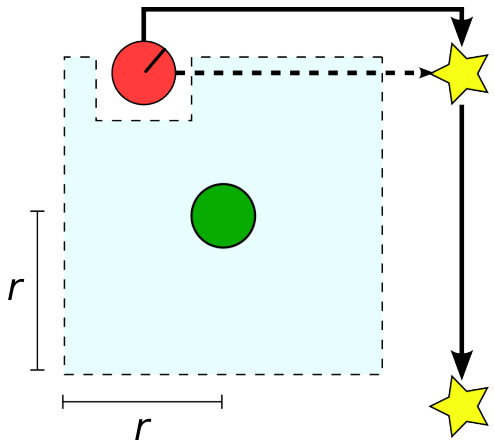

Fig. 1. Illustration of the presented covering box method. The green circle is a dynamic obstacle, the light blue region surrounding it is a virtual static obstacle. The robot (red circle) modifies its strategy as if the virtual obstacle was discovered online. It is attempting to reach the two yellow stars, as specified in its task formula φ.

I. Introduction

Any practical robot will face interaction with uncertain obstacles while attempting to complete its task. The nature of this uncertainty varies from the unknown, fixed layout of static (non-moving) obstacles, as addressed by vanilla occupancy grids [1], to the trajectories of goal-oriented, interacting crowds of people [2]. In the present work we are concerned with task completion in the midst of moving obstacles, where the task is specified formally in the GR(1) fragment of linear temporal logic, and the dynamic obstacles are assumed to have bounded speed. Such task specifications are known as *reactive synthesis* problems in the theoretical computer science literature [3], or *nondeterministic games against nature* in the planning literature [4]. An exact solution is required, meaning that any move by the adversary must be accounted for, and thus exponential complexity in the number of problem variables is unavoidable without further assumptions. The situation is no better in the continuous setting. Considering instead the continuous dynamics that are usually abstracted by a discrete game view, it has been shown that motion planning for a rigid body among moving obstacles is PSPACE-hard, even when the trajectories of these obstacles are known a priori [5].

Much previous work concerning formal methods in robotics includes a dynamic obstacle, where "dynamic obstacle" is taken to mean an uncontrolled object that can change its position in the workspace, and against which collisions must be avoided. Of course, "must" is relaxed to a probabilistic notion in some instances. Several examples involving straigtforward application of GR(1) synthesis in which the moving obstacle is explicitly encoded in the GR(1) task formula are given by Kress-Gazet et al. [6]. The limitations of explicit encoding of obstacles are addressed by a state space decomposition in the receding horizon approach of Wongpiromsarn (again, using GR(1) synthesis for strategy construction) [7]. In a problem involving tracking of nondeterministic moving targets, which is similar to the setting of dynamic obstacles, Özay and collaborators exploit parallel structure in the problem and propose a method for distributed synthesis to cope with a large discrete state-space [8].

In light of the computational complexity of the exact problem, a major family of alternatives is probabilistic methods. Restricting our attention to those involving a formal task specification, perhaps the work closest to the present is that of [9], in which the authors present a method that combines the strategy automaton from GR(1) synthesis with probabilistic obstacle tracking, the output of which is made suitable for direct input to the automaton by a safety threshold. A growing body of literature focuses on probabilistic model checking techniques and relies on synthesis for Markov decision processes. In [10], the authors cope with uncertainty about a dynamic environment by formal language learning. To avoid state explosion, Wongpiromsarn and collaborators propose a method that incrementally accounts for more states from the models of dynamic obstacles [11], improving the

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

solution depending on computational resources.

Our contributions are summarized as follows. In Section III, we present a substantial extension to the algorithm of [12] that allows strategy modification across goal nodes, thus obtaining a sort of monotonicity. In preparation for the algorithm, we also examine the structure of a graph derived from the strategy automaton that may be of independent interest for online formal synthesis algorithms in robotics. In Section IV, we present a method for collision avoidance with moving obstacles while completing a formally specified task. Our method is based on virtual static obstacles that change position slowly with time. Preliminary experiments validating the presented methods are described in Section V.

II. Preliminaries and problem formulation

We now describe the two basic components of a problem instance: the task formula φ and the dynamical model of moving obstacles. The task formula is expressed in the GR(1) fragment of linear temporal logic (LTL), which is an extension of Boolean (propositional) logic for countably infinite time sequences [13]. Recall that a Boolean logic formula can include "and" $\wedge$, "or" $\vee$, "negation" $\neg$, and "implication" $\implies$ operators, combined with atomic propositions that evaluate to either True or False. Temporal logic is useful for describing robot tasks because one can unambiguously express desired behaviors, e.g., recurring surveillance or responsiveness to external flags. While there are many temporal operators in the syntax of LTL, only "eventually" $\diamondsuit$ and "always" $\square$ are crucial in this paper. For a general introduction to LTL and relevant theory, consult e.g., [14]. An LTL formula φ that can be written as

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

is said to be a GR(1) formula [15], provided the subformulae have the following structure. Let $\mathcal{X}$ and $\mathcal{Y}$ be sets of variables, referred to respectively as "environment" and "system" variables. A *valuation* is an assignment of values to these variables, and the sets of all possible values for each are $\Sigma_{\mathcal{X}}$ and $\Sigma_{\mathcal{Y}}$, respectively. θ_{env} is a Boolean formula written in terms of $\mathcal{X} \cup \mathcal{Y}$ that specifies initial conditions. ρ_{env} is a Boolean formula in terms of $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X}$, where $\bigcirc \mathcal{X}$ denotes values of environment variables at the next time step. Thus, ρ_{env} is sometimes called the environment *transition rule* because it describes how the adversary may move, given a particular valuation of $\mathcal{X} \cup \mathcal{Y}$. The temporal operator $\square$ is written before ρ_{env} to indicate that it must hold at each time step, i.e., the transition rule always applies. Finally, the subformulae ψ_j^{env} are liveness conditions that the environment is assumed to satisfy infinitely often (hence the use of "$\square \diamondsuit$"). Notice the parallel structure in (1). θ_{sys} and $\psi_0^{\mathrm{sys}}, \ldots, \psi_{n-1}^{\mathrm{sys}}$ are analogous to the similarly named subformulae on the left-side of the implication. However, ρ_{sys} is predicated on $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X} \cup \bigcirc \mathcal{Y}$; again it can be regarded as a transition rule, but now it describes how the system (robot) may move, given the current valuation and the move that the environment is about to take. The left-side of the equation is often called the "assumption" because it restricts how the uncertain environment may behave, while the right-side is called the "guarantee" because it specifies behavior that is to be realized.

As a small example of a GR(1) formula, consider

$$\square \diamondsuit \texttt{door\_open} \implies \square \diamondsuit \texttt{visit\_bedroom}.$$

It has one environment variable, `door_open`, that is True whenever the door to a bedroom that is under surveillance is open. Given this door is always eventually open, expressed on the left-side of the formula, the robot is to recurringly visit the room, as indicated on the right-side of the formula. Of course, this is a description of the task, not its solution. More generally, given a GR(1) formula φ, one may algorithmically construct a finite automaton that accepts as input valuations of $\mathcal{X}$ and selects (outputs) valuations for $\mathcal{Y}$ such that the resulting sequence satisfies φ. A *finite automaton* $A = (V, \delta, L)$ is a triple where V is a set of nodes, $\delta \subseteq V \times \Sigma_{\mathcal{X}} \times V$ is a transition function, and $L : V \to \Sigma_{\mathcal{X}} \times \Sigma_{\mathcal{Y}}$ is a node labeling, or output function, of A. $(u, e, v) \in \delta$ is also denoted by $\delta(u, e) = v$. A finite automaton under which all sequences of valuations for $\mathcal{X} \cup \mathcal{Y}$ satisfy φ is said to be *winning*. A finite automaton that conforms with (obeys) the transition rules of φ is called a *strategy automaton*.

Given any Boolean formula ψ, the set of states from $\Sigma_{\mathcal{X}} \times \Sigma_{\mathcal{Y}}$ that satisfy it is denoted $[\![\psi]\!]$. An equivalent game graph can be obtained from the transition rules in a GR(1) formula φ, and the notation of $[\![\cdot]\!]$ can be extended to μ-calculus formulae concerning this graph. The reader is directed to [12] for an introduction relevant to our current setting and to [16] for a more general introduction.

In the context of a robot system architecture, the strategy automaton plays the role of the task-level controller, or what is occasionally named the finite-state machine. In addition to governing intrinsically discrete variables, such as whether a camera captures photographs, the strategy automaton selects movements in the physical workspace. The continuum of a workspace is mapped to a discrete representation, such as a cell decomposition [4], with adjacent cells labeled as connected if there exist controllers to steer between them. The relationship between navigation on a finite graph and in the actual workspace is made precise by bisimulations, sometimes called discrete abstractions [17]; for an introductory treatment, consult [18]. In this paper we tacitly assume the availability of an appropriate abstraction, allowing our treatment to be entirely in terms of discrete objects. A concrete example involving planar mobile robots appears in Section V.

The second major part of a problem instance is the dynamics model for the moving obstacles. In this paper we consider only random walks with single integrator dynamics. Each obstacle has the shape of a circle with radius γ. The set of points making up the body of the obstacle is called $\mathcal{O}$,

which we also use simply to refer to that obstacle; its center point is x. The moving obstacle has trajectories satisfying

$$\dot{x} = u, \qquad |u(t)| \leq 1, \tag{2}$$

where $x(t) \in \mathbb{R}^2$, $|\cdot|$ is the (Euclidean) 2-norm, and u is a piecewise constant function of time, changing at most once per second. The choice of unit bound on input in (2) is to simplify the presentation by normalizing with respect to maximum speed. Such a model is conservative because in general obstacles will not be able to move instantaneously in any direction. Futhermore, it is easy to see that (2) implies a reachability horizon for each time step. Sampling with period 1, the obstacle motion follows $x(t+1) = x(t) + u(t)$, hence

$$|x(t+1) - x(t)| \leq 1.$$

If the discretization of the workspace is appropriately chosen, then we can, possibly conservatively, model the obstacle as being able to move by at most one cell per time step. In this paper, while not required for the results, we assume an axis-aligned uniform grid partition of the plane for simplicity of presentation. The consequence of modeling the obstacle motion in this way is that our method can be ensured to work against obstacles with motion more constrained than (2). In terms of LTL, a random walking obstacle will eventually always ($\square\diamondsuit$) visit every cell, assuming the workspace is topologically connected. Thus we have obtained a fairness constraint that the obstacle will not block doorways, etc., without having to explicitly encode it into the task formula φ, as in previous work.

Summarizing the setting, we are given a workspace $\mathcal{W} \subseteq \mathbb{R}^2$, a finite partition of it, a task formula φ in the form of (1) specifying what the robot must do in $\mathcal{W}$, possibly in the presence of other discrete variables included in $\mathcal{X} \cup \mathcal{Y}$, and one or more moving obstacles $\mathcal{O}_1, \ldots, \mathcal{O}_k$ in $\mathcal{W}$.

III. Patching across goal states

Let φ be a task formula, and let $A = (V, \delta, L)$ be a winning strategy automaton for it. A limitation of the algorithm presented in [12] is that nodes in A corresponding to goal satisfaction cannot be removed. When some of the goals $\psi_0^{\text{sys}}, \ldots, \psi_{n-1}^{\text{sys}}$ depend on robot position in the workspace $\mathcal{W}$, it can easily occur that a wandering obstacle transiently covers goal states that would be reached under A. Nonetheless, the algorithm of [12] has been shown to successfully incorporate discovered positions of static obstacles, and thus we are motivated to extend it so as to be robust against more types of uncertain moving obstacles. Furthermore, our development here achieves a certain monotonicity with respect to approaching global re-synthesis as the portion of A that is mended increases.

To begin, define the edge set of a strategy automaton to be

$$E = \{(u, v) \in V \times V \mid \exists e \in \Sigma_{\mathcal{X}}, \delta(u, e) = v\}. \tag{3}$$

As an alternative to the input-oriented definition of δ, we may now view A as a directed graph (V, E). Recall from [12] that we may obtain during strategy synthesis a reach annotation

$$\text{RA} : V \to \{0, \ldots, n-1\} \times \mathbb{Z}$$

at no extra computational complexity, up to a constant factor. For any node $v \in V$, the first element $\text{RA}_1(v)$ is said to be the *goal mode* of v. Intuitively, upon reaching the node v, automaton A is pursuing a state that satisfies $\psi^{\text{sys}}_{\text{RA}_1(v)}$. Whether it is actually making progress toward that goal is indicated by the second element RA_2. If RA_2 decreases across an edge (u, v) in A (i.e., $\text{RA}_2(v) < \text{RA}_2(u)$), then strict progress is made. Otherwise one of the environment liveness conditions $\psi_0^{\text{env}}, \ldots, \psi_{m-1}^{\text{env}}$ in (1) is being blocked. When the goal ψ_i^{sys} of the current mode i being pursued is reached, the goal mode is incremented (modulo n) until an index j is found such that ψ_j^{sys} is not satisfied at the current state. Such a transition corresponds to an edge $(u, v) \in E$ where $\text{RA}_1(u) = i$ and $\text{RA}_1(v) = j$.

In order to see the structure of plays under A in terms of reaching goal states, first partition the set of nodes according to goal mode, i.e.,

$$V_i = \{v \in V \mid \text{RA}_1(v) = i\}, \tag{4}$$

and then define an edge set over this partition by

$$\hat{E} = \{(V_i, V_j) \mid \exists u \in V_i, \exists v \in V_j : (u, v) \in E\}. \tag{5}$$

Note that $\hat{E}$ is well-defined because $V_0, V_1, \ldots, V_{n-1}$ are mutually disjoint. From the description of reach annotation RA above, it should be clear that edges in $\hat{E}$ correspond to satisfaction of task goals $\psi_0^{\text{sys}}, \ldots, \psi_{n-1}^{\text{sys}}$ because across any such edge, the mode RA_1 changes.

It is natural to ask what structure the directed graph $\left(\{V_0, V_1, \ldots, V_{n-1}\}, \hat{E}\right)$ has, given its construction from (1) and a winning strategy automaton equipped with reach annotation RA. While exploring this further is the topic of future work, it turns out that this graph is a chain for many tasks in robotics, as we now show. By construction of RA, if the goal mode changes by more than 1, then more than one goal must be satisfied upon reaching that node during any play. A weaker result sufficient for our purposes is the following.

Remark 1: If there exists an edge $(u, v) \in E$ such that

$$|\text{RA}_1(v) - \text{RA}_1(u) \mod n| > 1,$$

then there exist goal indices i, j such that $[\![\psi_i^{\text{sys}}]\!] \cap [\![\psi_j^{\text{sys}}]\!] \neq \emptyset$.

Typically the goals in a robot task are disjoint. For a small example, consider surveillance of several locations in an office building; the robot cannot simultaneously occupy multiple locations at once, and therefore the corresponding goal conditions $\psi_0^{\text{sys}}, \ldots, \psi_{n-1}^{\text{sys}}$ in (1) are disjoint. This remains true even if discrete (non-locative) variables are introduced into the task formula φ, provided that position variables are combined with the other variables by conjunction, e.g., go to that room and capture a photograph. In Boolean logic notation, this property is expressed by

$$\psi_i^{\text{sys}} \wedge \psi_j^{\text{sys}} \equiv \text{False} \qquad \text{for all } i \neq j, \tag{6}$$

or in terms of sets of states, $[\![\psi_i^{\text{sys}}]\!] \cap [\![\psi_j^{\text{sys}}]\!] = \emptyset$ for all distinct pairs of indices i, j. This is exactly the contrapositive of Remark 1.

Therefore, for any robot task satisfying (6), it follows from Remark 1 that the graph $\left(\{V_0, V_1, \ldots, V_{n-1}\}, \hat{E}\right)$ can only have edges of the form (V_i, V_{i+1}) or (V_i, V_{i-1}), where index arithmetic is modulo n. Hence it is easy to see that any cycle either has length 2 or n. (It is immediate from the definition of $\hat{E}$ in (5) that there cannot be self-loops, i.e., cycles of length 1.)

Finally, it is common in GR(1) synthesis algorithms to construct a finite automaton that pursues each system goal in the order originally given in the task formula [19]. While there has been work addressing how reordering of these goals can improve performance in application to robotics [20], such reorderings can be made before invoking a synthesis algorithm (as a "black box"). Without loss of generality, we can assume that system goals are pursued in order, as appearing in the task formula φ. To see this, notice that mode i only concerns pursuit of a particular goal ψ_i^{sys}—satisfaction of a different goal ψ_j^{sys}, $j \neq i$, en route will not affect the current mode. Combining this with previous observations, the following lemma is proven.

Lemma 2: The graph $\left(\{V_0, V_1, \ldots, V_{n-1}\}, \hat{E}\right)$ is a subgraph of a chain.

Note that the lemma states "subgraph of a chain" because a winning strategy automaton can block one of the environment liveness conditions ψ_j^{env} of (1), possibly causing a node V_i to have no outgoing edges. A practical example is a strategy in which the robot blocks the doorway, thus preventing the door from closing as may have been assumed to always eventually happen.

We are now ready to present our extension to the automaton patching algorithm of [12]. Let $A = (V, \delta, L)$ be a strategy automaton that is winning with respect to the GR(1) task formula φ. Let $\bar{N} \subseteq \Sigma_{\mathcal{X}} \times \Sigma_{\mathcal{Y}}$ be a set of discrete states, e.g., corresponding to a ball in the robot workspace, over which the strategy from A must be changed. A change is required because a new task formula φ' was obtained due to modification of the transition rules (cf. (1)). Using the inverse of the node labelling function L, we can find sets of nodes that must be replaced and partition them according to goal mode. Precisely, for each $i = 0, 1, \ldots, n-1$, define

$$N_i = \left\{u \in V \mid L(u) \in \bar{N} \wedge \mathrm{RA}_1(u) = i\right\}.$$

Clearly, $N_i \subseteq V_i$, and indeed, this set is equivalently expressed by $N_i = L^{-1}(\bar{N}) \cap V_i$. Regarding the sets $N_0, N_1, \ldots, N_{n-1}$ as vertices in a graph, a new edge set $\hat{E}_N$ is constructed similarly to $\hat{E}$ in (5). Since each N_i is a subset of V_i, it is easy to see that an edge is in $\hat{E}_N$ only if it is in $\hat{E}$. Therefore it follows from Lemma 2 that the graph $\left(\{N_0, N_1, \ldots, N_{n-1}\}, \hat{E}_N\right)$ is also a subgraph of a chain.

Thus, there are three possibilities for each part of this graph, and we treat them separately. We now summarize the steps for modifying the automaton A to recover correctness with respect to the modified task formula φ' by changing behavior over the states in N. A formal statement is given in Algorithm 1, where we make use of a subroutine $\mathrm{Reach}_\varphi(A, B)$ that solves a reachibility game using the transition rules from φ: from any initial state in A, reach some state in B. This subroutine is straightforward to implement, and there are various methods for doing so. We assume it is done symbolically (i.e., in terms of sets of states) with a μ-calculus fixed point iteration similar to that in [12], to which the reader is referred for details. That is, $\mathrm{Reach}_\varphi(A, B)$ is a μ-calculus formula, and thus $[\![\mathrm{Reach}_\varphi(A, B)]\!]$ denotes the fixed point set of states. Whatever reachability technique is chosen, the crucial properties are that $[\![\mathrm{Reach}_\varphi(A, B)]\!] \subseteq A$ and from any initial state in $[\![\mathrm{Reach}_\varphi(A, B)]\!]$, there exists a strategy to drive the play to a state in B, or to block one of the environment liveness conditions in φ.

Before examining the three possibilities for $\left(\{N_0, N_1, \ldots, N_{n-1}\}, \hat{E}_N\right)$, observe that any edge in $\hat{E}_N$ corresponds to a robot goal being reached, i.e., (N_i, N_{i+1}) corresponds to an action by the automaton A in which a ψ_i^{sys}-state is visited and the goal mode is incremented, modulo n. The first possibility is that this graph is itself a chain; this is equivalently stated as the case where there is transition into a goal state for each of the goals. Motivated by practical rarity, in this case we simply perform global re-synthesis of the strategy automaton A for the modified task formula φ'.

For the remaining two possibilities, the graph $\left(\{N_0, N_1, \ldots, N_{n-1}\}, \hat{E}_N\right)$ is not itself a chain. However, recall that it is a subgraph of a chain, and therefore each of the sets N_i is either isolated (i.e., without ingoing or outgoing edges) or part of a finite sequence of edges of the form $(N_{i-1}, N_i), \ldots, (N_i, N_{i+1})$. An example of this is given in Figure 2. In the former case, the nodes of A in N_i can be replaced without changing the goal mode i. This is achieved in two steps. First, find all nodes in N_i that have an ingoing edge in E (recall (3)) from a node outside N_i, and call the set of state labels of these nodes Entry^i. (Recall that state labels of automaton nodes are provided by L, and N_i is a set of nodes.) Similarly, find all nodes in N_i that have an edge going out of N_i, and call the result Exit^i. Note that $\mathrm{Entry}^i, \mathrm{Exit}^i \subseteq \Sigma_{\mathcal{X}} \times \Sigma_{\mathcal{Y}}$. A substrategy suitable for patching A is found by solving $\mathrm{Reach}(\mathrm{Entry}^i, \mathrm{Exit}^i)$.

In the latter case, let I be a subset of $\{0, 1, \ldots, n-1\}$ indexing the nodes N_i that form a finite sequence of edges in $\hat{E}_N$. We create a new substrategy by solving a sequence of reachability games. To begin, let the initial index in I be i, so that $I = \{i, i+1, \ldots, i+(|I|-1)\}$, where index arithmetic is modulo n. The sets of nodes Entry^I and Exit^I are created in a manner similar to the previous case, except that $\mathrm{Entry}^I \subseteq L(N_i)$ and $\mathrm{Exit}^I \subseteq L(N_{i+(|I|-1)})$. Beginning at the end of this sequence, first solve $\mathrm{Reach}_\varphi([\![\psi^{\text{sys}}_{i+(|I|-1)-1}]\!], \mathrm{Exit}^I)$. This method terminates either when $[\![\psi^{\text{sys}}_{i+(|I|-1)-1}]\!]$ is obtained, or when a fixed point occurs. In the former case, let $B_{i+(|I|-1)-1} := [\![\psi^{\text{sys}}_{i+(|I|-1)-1}]\!]$, and in the latter case, let $B_{i+(|I|-1)-1}$ be the intersection of the fixed point with $[\![\psi^{\text{sys}}_{i+(|I|-1)-1}]\!]$. Note that, having been obtained from

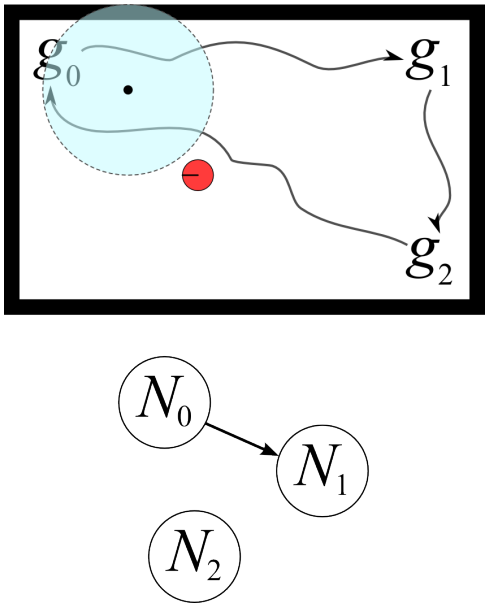

Fig. 2. Example of the graph $\left(\{N_0, N_1, N_2\}, \hat{E}_N\right)$ of sets of affected nodes, over which patching will occur. The workspace is shown above this graph, with a light blue circle indicating states over which re-planning should occur. In this case, there are three system goals: $\psi_0^{\text{sys}}, \psi_1^{\text{sys}}, \psi_2^{\text{sys}}$, which correspond to positions g_0, g_1, and g_2, respectively, in the workspace. Note that the gray curves indicate *possible* paths, whereas A is actually a *strategy* and thus could lead to many different paths. Nodes in A that correspond to pursuit of ψ_2^{sys}-states are not affected by the change set N in this example. Hence, $N_2 = \emptyset$, and the N_2 vertex is isolated in the above figure. By contrast, there is an edge from N_0 to N_1, indicating that a goal state is reached by an edge affected by the change set N. Thus, in terms of Algorithm 1, the index set I_0 will have two elements, and the substrategy corresponding to that edge will reach a ψ_2^{sys}-state en route to the Exit set.

Algorithm 1 Patching with goal states

INPUT: GR(1) formula φ, strategy A, reach annotation RA, modified formula φ', neighborhood $\bar{N} \subseteq \Sigma_{\mathcal{X}} \times \Sigma_{\mathcal{Y}}$
OUTPUT: set of tuples $(A^j, \text{Entry}^j, \text{Exit}^j, I_j)$ with partial reach annotation RA^j
$\texttt{Patches} := \emptyset$
$N := L^{-1}(\bar{N})$
for all $i = 0, 1, \ldots, n-1$ //Sort by goal mode **do**
  $N_i := \{u \in N \mid \text{RA}_1(u) = i\}$
end for
$\hat{E}_N := \{(N_i, N_j) \mid N_i \times N_j \cap E \neq \emptyset\}$
$\kappa := 0$
$I_0 := \{0\}$
for all $i = 0, 1, \ldots, n-1$ //Partition indices **do**
  if $(N_i, N_{i+1 \mod n}) \in \hat{E}_N$ **then**
    $I_\kappa := I_\kappa \cup \{i+1 \mod n\}$
    if $i = n-1$ **then**
      $I_0 := I_0 \cup I_\kappa$ //Merge first and last index sets
      $\kappa := \kappa - 1$
    end if
  else
    $\kappa := \kappa + 1$
  end if
end for
for all $j = 0, 1, \ldots, \kappa$ **do**
  $B := \text{Exit}^j$
  Set A^j to `nil`
  Set i such that $I_j = \{i, i+1, \ldots, i+|I_j|-1\} \mod n$.
  for all $\texttt{offset} = |I_j|-2, |I_j|-3, \ldots, 0$ **do**
    $C := [\![\text{Reach}_{\varphi'}([\![\psi^{\text{sys}}_{i+\texttt{offset}}]\!], B)]\!]$
    if $C = \emptyset$ **then**
      abort //Failed to find a substrategy
    end if
    Compute strategy $A^j_{C \to B}$ to reach B from C.
    Merge $A^j_{C \to B}$ into A^j.
    $B := C$ //Prepare for next loop iteration
  end for
  $C := [\![\text{Reach}_{\varphi'}(\text{Entry}^j, B)]\!]$
  if $\text{Entry}^j \not\subseteq C$ **then**
    abort //Failed to find a substrategy
  end if
  Compute $A^j_{\text{Entry}^j \to B}$ to reach B from Entry^j.
  Merge $A^j_{\text{Entry}^j \to B}$ into A^j.
  $\texttt{Patches} := \texttt{Patches} \cup (A^j, \text{Entry}^j, \text{Exit}^j, I_j)$
end for
return `Patches`

$\text{Reach}()$, it is possible to reach Exit^I from any initial state in $B_{i+(|I|-1)-1}$. Now, another reachability game is solved: $\text{Reach}_\varphi([\![\psi^{\text{sys}}_{i+(|I|-1)-2}]\!], B_{i+(|I|-1)-1})$. As in the previous step, store the result to $B_{i+(|I|-1)-2}$. This process is repeated until B_i is computed. If $\text{Entry}^I \subseteq B_i$, then strategies for each of the reachability games are chained together so that the resulting strategy automaton can drive the actual game, which is governed by transition rules in φ, from any initial state in Entry^I to some state in Exit^I, while visiting robot goals $\psi_i^{\text{sys}}, \psi_{i+1}^{\text{sys}}, \ldots, \psi_{i-(|I|-1)-1}^{\text{sys}}$ en route. Otherwise, the algorithm aborts with failure.

In terms of Algorithm 1, if substrategy construction succeeds for each element I_κ of the partition of indices, then the resulting substrategies in `Patches` can be patched into the original automaton A' in an entirely similar manner as for singleton I_κ in the algorithm of [12].

IV. Bounding Boxes for Moving Obstacles

The original motivating context is completion of a high-level task in the midst of moving obstacles. As described in Section I, inclusion of all possible moves by a dynamic obstacle leads to exponential explosion in the number of discrete states, and solving classical motion planning among moving obstacles of known trajectories in a continuous setting is PSPACE-hard. GR(1) synthesis, being an exact game graph solution method, is thus impractical for dynamic obstacle avoidance. Instead of explicitly encoding possible obstacle movement as an environment variable in the task formula φ, we treat it separately as follows.

The basic idea of our approach is to cover the dynamic obstacle with a larger, virtual static obstacle. Changes to the static layout of the workspace are easier to treat because, in terms of the discrete abstraction, they only affect reachability of robot states $\Sigma_{\mathcal{Y}}$ and are easily encoded by changing the

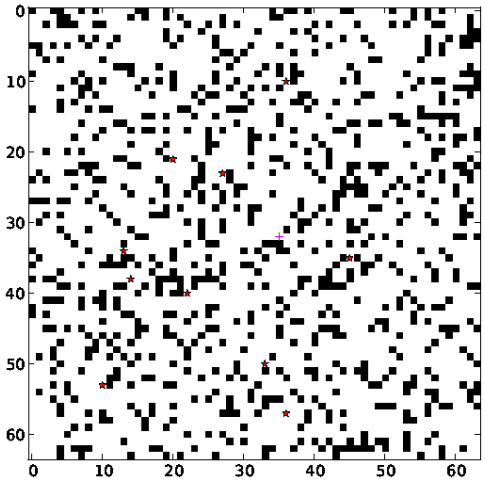

Fig. 3. Random gridworld of size 64×64 with 0.2 block density. 10 positions for surveillance are indicated by small red stars, and the initial robot position is magenta cross (near the center).

transition rules ρ_{sys} in the task formula φ (recall (1)). This setting is illustrated in Figure 1, where the green dynamic obstacle is covered by a square with side length $2r$.

When the dynamic obstacle is first detected, a virtual static obstacle is created to cover it. If this virtual obstacle would also cover the robot, then a notch is made so that the strategy automaton, once updated, will direct motion away.

V. Simulation experiments

The algorithm presented in Section III was implemented in the C programming language as part of `gr1c`[1]. Preliminary simulation experiments were conducted using large random layout gridworlds generated with `TuLiP`[2], the latter also providing a Python interface to the patching routines in `gr1c`. All of this will soon be released open source.

An example of a randomly generated gridworld is shown in Figure 3. The test setting consists of a nominal strategy in which the robot infinitely often visits all goal positions and a random walking obstacle that can move by at most one cell per time step.

VI. Future work

Despite now being able to patch any part of a strategy automaton, rather than only nodes in-between goals, the algorithm is still not complete. That is, it can abort and "fall back" on global re-synthesis even when a local solution exists. Future work will address whether incompleteness is unavoidable, and if so in the general case (likely), whether this limitation carries over to discrete abstractions obtained from manifolds relevant for robotics, e.g., $\mathbb{R}^2$ and $\mathrm{SE}(2)$.

A second topic of future work is treatment of other moving obstacle dynamics. For instance, an obstacle constrained by unicycle dynamics and bounded inputs could allow a virtual static obstacle that is narrow compared with those found in this paper.

[1] `http://scottman.net/2012/gr1c`
[2] `http://tulip-control.sf.net`

Acknowledgments

This work was partially supported by the Boeing Corporation.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2006.

[2] P. Trautman, "Robot navigation in dense crowds: Statistical models and experimental studies of human robot cooperation," Ph.D. dissertation, California Institute of Technology, Pasadena, California, USA, 2012. [Online]. Available: http://resolver.caltech.edu/CaltechTHESIS:05182013-191132413

[3] 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.

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

[5] J. Reif and M. Sharir, "Motion planning in the presence of moving obstacles," in *Foundations of Computer Science, 26th Annual Symposium on*, October 1985, pp. 144–154.

[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] T. Wongpiromsarn, "Formal methods for design and verification of embedded control systems: Application to an autonomous vehicle," Ph.D. dissertation, California Institute of Technology, 2010. [Online]. Available: http://resolver.caltech.edu/CaltechTHESIS:05272010-153304667

[8] N. Ozay, U. Topcu, T. Wongpiromsarn, and R. M. Murray, "Distributed synthesis of control protocols for smart camera networks," in *International Conference on Cyber-physical Systems*, 2011.

[9] 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)*, Saint Paul, Minnesota, USA, May 2012, pp. 330–337.

[10] Y. Chen, J. Tumová, and C. Belta, "LTL robot motion control based on automata learning of environmental dynamics," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, Minnesota, USA, May 2012, pp. 5177–5182.

[11] T. Wongpiromsarn, A. Ulusoy, C. Belta, E. Frazzoli, and D. Rus, "Incremental temporal logic synthesis of control policies for robots interacting with dynamic agents," Tech. Rep., March 2012. [Online]. Available: arxiv.org/abs/1203.1180

[12] 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)*, Karlsruhe, Germany, May 2013, pp. 4573–4580.

[13] E. A. Emerson, *Handbook of theoretical computer science (vol. B): formal models and semantics*. MIT Press, 1990, ch. Temporal and modal logic, pp. 995–1072.

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

[15] 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.

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

[17] 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.

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

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

[20] G. Jing and H. Kress-Gazit, "Improving the continuous execution of reactive LTL-based controllers," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013, pp. 5419–5425.