Patching task-level robot controllers based on a local $\mu$-calculus formula

Scott C. Livingston  
Pavithra Prabhakar  
Alex B. Jose  
Richard M. Murray

Abstract—We present a method for mending strategies for GR(1) specifications. Given the addition or removal of edges from the game graph describing a problem (essentially transition rules in a GR(1) specification), we apply a $\mu$-calculus formula to a neighborhood of states to obtain a "local strategy" that navigates around the invalidated parts of an original synthesized strategy. Our method may thus avoid global re-synthesis while recovering correctness with respect to the new specification. We illustrate the results both in simulation and on physical hardware for a planar robot surveillance task.

I. INTRODUCTION

A major research theme of the last two decades is to bring verification and synthesis methods to bear on problems in hybrid systems. For example, see Davoren and Nerode [1] and other papers in that special issue of Proceedings of the IEEE; an early example of theorem-proving based verification for control problems is [2]. However, one quickly arrives at undecidable problems (e.g., see [3], [4] and references therein). Thus a current topic is to find interesting classes of hybrid control systems that admit tractable synthesis algorithms—at least in appropriate cases. A motivation for the present work is to exploit additional structure not available in the classical automata-theoretic formulation of reactive synthesis.

Perhaps the simplest structure provided by hybrid systems is a notion of “localness.” Given a vertex in a discrete transition system, how can we find other vertices that are sufficiently similar? If the set of vertices maps into a metric space, then immediately we can look at a ball centered on the given vertex to form a “neighborhood of vertices.” This is just a basic question of how to cluster data and arises in many contexts. Similarly, a metric may be provided directly for the nodes of an automaton, e.g., as used to formulate robustness in [5].

In the present work, we are concerned with using localness to estimate what parts of a control strategy must be updated in response to a game change. In particular we consider application of these methods in mobile planar robotics. A typical problem in this setting is updating a map describing the locations of obstacles and other areas of interest, such as a charging station. One representation often used is cell decompositions [6], which may be polygonal partitions of part of the state space (such as position but not velocity).

If appropriate properties relating the continuous dynamics to discrete movement among cells are satisfied [7] (essentially so that a bisimulation is obtained), then it suffices to solve the synthesis problem over this discrete abstraction. Forming discrete abstractions in this manner and then synthesizing game strategies on the resulting game graph forms the basis of some previous work in robotics [8], [9]. While this situation is superficially very attractive, the number of cells required to solve a problem may be large, and when combined with uncontrolled environment behavior, the cardinality of the discrete abstraction may be such that we have gained little from the abstraction process.

Given that a global strategy may be hard to construct and fragile in settings with inherent uncertainty, one may ask whether local parts of the strategy can be updated as new data is accumulated online. That is, can we perform incremental synthesis by working with neighborhoods of nodes based on metrics from the underlying continuous space?

In [10], Livingston et al. address a similar problem as in the present work. However, they treat the synthesis algorithm as a “black box” to be invoked with local specifications, which are then used to mend the original strategy. Here we explicitly propose a $\mu$-calculus formula with which to synthesize a local strategy.

II. SYNTHESIS PROBLEM

A. GR(1) specifications and associated game

For convenience, several LTL operators are listed in Table I. For an introduction to relevant theory see, e.g., [11]. Let $\mathcal{X}$ be a set of uncontrolled variables, and $\mathcal{Y}$ a set of controlled variables. Each variable has a finite domain, and the product of these domains gives the discrete state set $\Gamma$. Restrictions to domains of uncontrolled and controlled variables are denoted by $\Gamma_X$ and $\Gamma_Y$, respectively. Denote the projection of a state $s \in \Gamma$ onto the variables in $\mathcal{X}$ by $s \downarrow_\mathcal{X}$. For a set $B \subseteq \Gamma$, projection is elementwise and denoted by $B \downarrow_\mathcal{X}$. Hence to be precise, $\Gamma_X := \Gamma \downarrow_\mathcal{X}$ and $\Gamma_Y := \Gamma \downarrow_\mathcal{Y}$. A state taken directly from a restricted domain will be marked with an appropriate subscript, e.g., $s_X \in \Gamma_X$. Compositions of these to form a complete state will be written as tuples, e.g., $(s_X, s_Y) \in \Gamma$. As in the literature, we also refer to $\mathcal{X}$ as “environment variables” and $\mathcal{Y}$ as “system variables.”

As defined in [12], a GR(1) (Generalized Reactivity of rank 1) specification is of the form

$$\theta_\varepsilon \land \square \rho_\varepsilon \land \left( \bigwedge_{j=0}^{m-1} \Box \diamond J_j^s \right) \implies \theta_s \land \square \rho_s \land \left( \bigwedge_{i=0}^{n-1} \Box \diamond J_i^s \right)$$

(1)

where $\theta_\varepsilon$, $\theta_s$, $J_0^s$, $\ldots$, $J_{m-1}^s$, and $J_0^s$, $\ldots$, $J_{n-1}^s$ are state formulae on $\mathcal{X} \cup \mathcal{Y}$, $\rho_\varepsilon$ is a state formula on $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}$, and $\rho_s$ is a state formula on $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X} \cup \mathcal{Y}$. Application of the “next” operator $\diamond$ to a variable denotes the value taken...
by that variable at the next time step. E.g., $x \rightarrow \bigcirc y$ is a
transition rule requiring that from any state where $x$ holds, $y$
must hold at the next time step.

The problem setting is best viewed as a game between the
environment, who sets variables in $X$, and the system, who
sets variables in $Y$. (See the definition of simulation game
structure in [12].) The transition rules $p_e$ and $p_s$ dictate how
the environment may move from a particular state, and how
the system may then respond. The allowed moves can be
expressed in a graph like structure called the game graph.

Given a GR(1) formula $\varphi$ as in equation (1), a game graph
up corresponding to it, denoted $G_\varphi$ is given by $(\Gamma, E_e, E_s)$, where

- $E_e(s) := \{ a_X \in \Gamma_X \mid (s, a_X) \models p_e \}$,
- $E_s(s, a_X) := \{ a_Y \in \Gamma_Y \mid (s, a_X, a_Y) \models p_s \}$.

An execution (or play) between the uncontrolled envi-
riment and controlled system corresponds to a path on
the game graph, beginning at a state satisfying the initial
conditions $\theta_e$ and $\theta_s$. A strategy on a game graph specifies
for each path in the system and an environment move, a
system move enabled at that point. We say that a strategy is
winning for a GR(1) formula, if every path which conforms
to the strategy satisfies all system goals or blocks at least one
environment goal. It is a well-known result that for GR(1)
specifications, there exists a finite memory winning strategy.

Hence, we define a strategy automaton.

A strategy automaton for a GR(1) formula $\varphi$ is a triple
$A = (V, \delta, L)$, such that $V$ is a set of nodes, $L : V \rightarrow \Gamma$, and
$\delta : (\{v \in V \mid X(v) \} \times E_e(L(v))) \rightarrow V$ such that for every $(v, s_X)$,
$L(\delta(v, s_X)) \models \varphi$ and $L(\delta(v, s_X)) \models \varphi$. We use $u \rightarrow_\delta v$
to denote there exists an environment move from state $L_1(u)$
such that automaton $A$ transitions to node $v$ in response.

An execution of a strategy automaton is a sequence
$v_1 v_2 \ldots v_n$ such that $v_i \rightarrow_\delta v_{i+1}$. A trace of a strategy
automaton is a sequence $s_1 s_2 \ldots s_n$ such that there exists
and execution $v_1 v_2 \ldots v_n$ with $s_i = L(v_i)$. We say that a
strategy automaton for $\varphi$ is winning from a set of states $I$
if $I \subseteq L(V)$, and every trace of the automaton satisfies the
environmental and system goals of $\varphi$, that is, either there
exists an environment goal $J^s_i$ which is not satisfied starting
from a certain point in the trace, or every system goal $J^s_i$ is
satisfied infinitely often in the trace. We simply say that
a strategy automaton is winning for $\varphi$ if $I$ contains the set of
states satisfying $\theta_e$ and $\theta_s$.

B. GR(1) synthesis algorithm

We present a method for synthesizing a strategy automaton
for a GR(1) formula. We first define a predecessor operator.
For a given set of states $X \subseteq \Gamma$, the predecessor operator
$Pre$ returns all states from which, given any possible envi-
noment move, there is some system move going to $X$. The
operation is a well known primitive in most algorithms for
reachability computations on game graphs or in other control
problems with disturbances. It is also sometimes written as
$circ$. Precisely, let $X \subseteq \Gamma$. Define

$$Pre_\varphi(X) := \{ s \in \Gamma \mid \forall a_X \in E_e(s),
\exists a_Y \in E_s(s, a_X) : (a_X, a_Y) \in X \}. \quad (2)$$

We now sketch one method to synthesize a strategy
for a GR(1) specification, e.g., as described in [13]. The
present paper only addresses strategies constructed in this
manner. Considering only such strategies is without loss of
generality because a realizable specification always admits
this synthesis method. For each goal $J^s_i$, $i \in \{0, \ldots, n-1\}$,
the basic approach during synthesis is to find at each step,
a move taking the game strictly closer to a $J^s_i$-state (a state
satisfying $J^s_i$) or to move in such a way that one of the
environment liveness conditions $J^e_j$, $j \in \{0, \ldots, m-1\}$, is
blocked. To be more precise, consider a row of the $\mu$-calculus
formula presented in [12] to compute the winning set $W_\varphi$
for a specification $\varphi$.

$$\mu Y \left( \bigvee_{j=0}^{m-1} \nu X \left( (J^s_i \land Pre_\varphi(Z_{i+1})) \right) \right) \land Pre_\varphi(Y) \lor \left( \neg J^s_i \land Pre_\varphi(X) \right) \quad (3)$$

where subscript addition is modulo $n$. For fixed values of
$Z_i$, the $\mu Y$ operators are used to compute the sets which
reach corresponding systems goals. Let us denote the $Y$ sets
obtained in the consecutive iterations for goal $i$ by $Y^0_i, Y^1_i, \ldots$.
These sets have the property that $Y^0_i \subseteq Y^1_i \subseteq \cdots \subseteq Y^n_i$,
and due to the finiteness of the state space, $Y^k_i = Y^{k+1}_i$ for
some $k$. Let $k$ be the minimal integer for which $Y^k_i = Y^{k+1}_i$.
Then, it also ensures that there is a strategy that
“progresses” towards the goal. For every state in $Y^k_i$, $k \geq 1$, the
strategy ensures that for any environment move, there is
a system move which results in a state $Y^{k'}_i$ for some
$k' < k$, or remains in some state $Y^k_i$, but violates one of the
environment goals, that is the current and next state both
violate a particular environment goal. Finally, when the $Z_i$’s
reach the fixed points (cf. [12]), every system goal that does
not eventually globally violate an environment goal has a
strategy to move to a state in some $Y^k_i$.

In order to propose a patching algorithm, we need to
assume that the initial strategy automaton corresponding to
the unmodified GR(1) formula has more information than
as defined previously. We use the above observations of
the synthesis algorithm to propose a data structure on the strategy
automaton, which gives an alternate characterization for the
existence of a winning strategy for the GR(1) formula.
Definition 1: A reach annotation on a strategy automaton $A = (V, \delta, L)$ for a GR(1) formula $\varphi$ is a function $RA : V \to [n] \times \mathbb{Z}_+$ which satisfies the following conditions. Define $RA_i$ for $i = 1, 2$ as $RA_i(v) = n_i$, where $RA(v) = (n_1, n_2)$, and let $[n] = \{0, 1, \ldots, n - 1\}$. Given $i < j$, the numbers between $i$ and $j$ are $i + 1, \ldots, j - 1$, and if $j \leq i$, then the numbers between $i$ and $j$ are $i + 1, \ldots, n - 1, 0, \ldots, j - 1$.

1) $RA_2(v) = 0$ iff $L(v)$ is a $RA_2(v)$-system goal.
2) For each $v \in V$ and $s_x \in E_x(L(v))$, if $RA_2(v) \neq 0$, then $RA_2(v) = RA_2(\delta(v, s_x))$ and either
   a) there exists an environment goal $J_p^0$ such that both $L(v)$ and $\delta(v, s_x)$ do not satisfy $J_p^0$; or
   b) $RA_2(v) > RA_2(\delta(v, s_x))$.
3) For each $v \in V$ and $s_x \in E_x(L(v))$, if $RA_2(v) = 0$, then there exists a $j$ such that for all $k$ between $RA_1(v)$ and $j$, $L(v)$ is $k$-system goal, and $RA_1(\delta(v, s_x)) = j$.

Remark 2: Note that one can easily extract a strategy automaton and a reach annotation from the $Y_i^j$ sets defined in the strategy synthesis algorithm above.

The next theorem provides a characterization of a winning strategy in terms of the reach annotation.

Theorem 3: There exists a winning strategy automaton $A$ for a GR(1) formula $\varphi$ if and only if there exists a strategy automaton with all initial states for $\varphi$ and with a reach annotation for $A$.

Proof: The "only if" part follows from the above remark. Conversely, let $A$ be a strategy automaton and $RA$ be a reach annotation on $A$. Let $s_0$ be an initial state for $\varphi$. By hypothesis there is a node $v_0$ in $A$ with $L(v_0) = s_0$. Let $v_0v_1v_2\cdots$ be an infinite execution originating at $v_0$. Note that by definition of strategy automata, such an infinite execution will occur if the environment moves are always possible (i.e., in $E_x(L(v_i))$ for each $v_i$). Otherwise, halting on a finite execution means that the environment has violated the assumption part of the GR(1) formula $\psi$ and therefore the resulting trace is winning. There are three possibilities for the execution $v_0v_1v_2\cdots$.

1) First, there is some $k$ such that $RA_2(v_i) = RA_2(v_{i+1}) \neq 0$ for $i \geq k$. Then by definition of reach annotation one of the environment goals is indefinitely violated and therefore the corresponding trace is winning.
2) Second, there is some $k$ such that $RA_2(v_i) = RA_2(v_{i+1}) = 0$. Then all system goals are satisfied at each step of the trace after $k$, and therefore the trace is winning.
3) Finally, there are infinitely many $i$ where $RA_2(v_i) \neq RA_2(v_{i+1})$. If there exists $k$ such that $RA_2(v_i) \leq RA_2(v_{i+1})$ for $i \geq k$, then because there are infinitely many indices where this inequality is strict, it must be that an environment goal is violated at $v_i$ onward and therefore the corresponding trace is winning. Otherwise, there are infinitely many $i$ where $RA_2(v_i) > RA_2(v_{i+1})$. Because $RA_2(V)$ is bounded below by 0, it follows that there are infinitely many indices $i$ where $RA_2(v_i) = 0$. Then, the third property of reach annotation ensures that all system goals are visited, and therefore the corresponding trace is winning.

III. PATCHING ALGORITHM

In this section, we propose an algorithm for patching a strategy or controller when there is a change in the edge set associated with a game graph. Let us fix a GR(1) formula $\varphi$ and a strategy automaton $A = (V, \delta, L)$ for $\varphi$ with a reach annotation $RA$ for the rest of the section.

A. Game changes

Suppose that the edges of the game graph associated with the GR(1) formula $\varphi$ change from $E_a$ and $E_a'$ to $E_a''$ and $E_a'''$, respectively. This affects a winning strategy automaton $A$ for $\varphi$ in two ways:

1) (removal of outgoing edges from controlled vertices)
Some control decisions can become unavailable. Let $v$ be a node in $A$ affected in this manner. Then:

$$\text{Cond}_1(v) := \exists s_x : (L(\delta(v, s_x)) \nexists E_a''(L(v), s_x).$$

2) (addition of outgoing edges to uncontrolled vertices)
The environment has new moves. A node $v$ is affected in this manner if:

$$\text{Cond}_2(v) := E_a'(L(v)) \nexists E_a''(L(v)) \neq 0,$$ (5)

The set of all nodes $v$ satisfying $\text{Cond}_1$ or $\text{Cond}_2$ are said to be the affected nodes of $A$ due to the changes $E_a'$ and $E_a''$ to the edges $E_a$ and $E_a'$ of the game graph of $\varphi$.

B. Overview

Consider the case where parts of the specification abstract the behavior of a dynamical control system. The precise meaning of “abstraction” is beyond the scope of this paper; we refer the reader to [7] and references therein for details. Suffice it to observe that, while the abstraction contributes variables (i.e., elements of $X' \cup Y'$) and transition rules (i.e., conjuncts in $\rho_v$ or $\rho_v$; recall that for state formulae $f$ and $g$, we have that $\square(f \land g) \equiv (\square f) \land (\square g)$) in a transparent manner, the underlying dynamics provide additional structure we hope to exploit. To that end, we assume we can compute a set of vertices in the game graph $(\Gamma, E_a, E_a')$ that is “local” in some sense. For instance, this could be obtained by computing the norm ball centered at some state of the underlying dynamical system and finding all vertices in $\Gamma$ corresponding to states contained in that norm ball [10]. We call this set $\hat{N}$ (recall $\hat{N} \subseteq \Gamma$) and refer to it as the “neighborhood of vertices.” This terminology is motivated by how we construct $\hat{N}$ rather than a topological space defined for the game graph $(\Gamma, E_a, E_a')$. The precise conditions under which $\hat{N}$ possesses desirable properties, such as being a small subset of $\Gamma$ or preserving local connectivity of the graph $(\Gamma, E_a, E_a')$, is the subject of future work.

Note that the reach annotation $RA$ defines a partition of the nodes in $A$ such that the nodes with $RA_1(v) = i$ correspond to those reaching system goal $i$ or eventually
globally violating an environment goal. Upon reaching a state satisfying \( J^*_e \), the goal mode is incremented, modulo \( n \), and the process continues. (Note that a state may satisfy more than one system goal.) The broad idea of the patching algorithm is to identify the affected nodes corresponding to each system goal and locally modify the automaton.

Suppose that the set of affected nodes due to the edge changes is non-empty. Let us denote the modified formula as \( \varphi' \), that is, the formula is the same as \( \varphi \), except that the \( \rho_e \) and \( \rho_s \) are replaced by \( \rho'_e \) and \( \rho'_s \) corresponding to edge changes in \( E_e \) and \( E_s \). For each goal mode \( i \), let \( U_i \) be the set of nodes in the strategy automaton affected by the change, which are not themselves labeled with a goal state. More precisely,

\[
U_i := \{ v \in V \mid L(v) \not\models J^*_i \wedge (\text{RA}_1(v) = i) \\
\wedge (\text{Cond}_1(v) \vee \text{Cond}_2(v)) \}. \tag{6}
\]

Observe that these sets are disjoint, and that one or more of them may be empty. Recall that nodes labeled with the same state have different goal modes. This is assumed without loss of generality because otherwise the strategy is redundant (not minimal).

Next, we define an operator for producing local strategies. Let \( \bar{N} \) and \( B \subseteq \Gamma \) be sets of states. Define a \( \mu \)-calculus formula

\[
\text{Local}(B, \bar{N}) := \mu \left( \bigvee_{j=0}^{m-1} \nu X \left( B \vee \left( \text{Pre}_{\varphi'}(Y) \wedge \bar{N} \right) \right) \wedge \left( \neg J^*_j \wedge \text{Pre}_{\varphi'}(X) \wedge \bar{N} \right) \right). \tag{7}
\]

This formula can be viewed as a truncated form of that presented in [12] (also see equation (3)) to compute the winning set for a GR(1) specification. Notice that states returned by the \( \text{Pre} \) operation are restricted to \( \bar{N} \). Let us denote the set of states in the fixed point computation of \( \text{Local}(B, \bar{N}) \) as \( \text{[Local}(B, \bar{N})] \). The result is a strategy which ensures that either some state in \( B \) is eventually reached or some environment goal is eventually persistently violated. Let us denote by \( (A', S_1, S_2) \) the fact that \( A' \) is a strategy automaton for \( \varphi' \), which is winning from any state in \( S_1 \) with respect to reaching some state in \( S_2 \), or eventually globally violating an environmental goal of \( \varphi' \) (same as that of \( \varphi \)). Note that there exists a \( A' \) such that \( (A', \text{[Local}(B, \bar{N})], B) \). It can be computed using the fixed point algorithm defined by the formula \( \text{Local}(B, \bar{N}) \).

Further, we can ensure the existence of the following partial reach annotation.

**Theorem 4:** Given \( (A', C, B) \), where \( A' = (V, \delta, L) \) and \( C \subseteq \text{[Local}(B, \bar{N})] \), there exists a partial reach annotation \( \text{RA} : V \to \{i\} \times \mathbb{Z}^+ \), for any \( i \) such that the following hold:

1) \( \text{RA}_2(v) = 0 \) iff \( L(v) \) is in \( B \).
2) For each \( v \in V \) and \( s_X \in \text{Ev}(v) \), if \( \text{RA}_2(v) \neq 0 \), then either
   a) there exists an environment goal \( J^*_e \) such that both \( L(v) \) and \( \delta(v, s_X) \) do not satisfy \( J^*_e \), or
   b) \( \text{RA}_2(v) > \text{RA}_2(\delta(v, s_X)) \).

**Proof:** This is immediate from the definition of the \( \mu \)-calculus formula and using the same construction from intermediate values of the fixed point computation of \( \text{[Local}(B, \bar{N})] \), as in Remark 2.

**Remark 5:** It is obvious from the proof that the above theorem is constructive, that is, one can compute a partial reach annotation for any \( (A', \text{[Local}(B, \bar{N})], B) \).

We assume at least one of the states in the automaton satisfying each goal remains feasible, i.e., there is some way to drive the play to it after the edge set change. In other words, we assume that we do not need to introduce new goal nodes into the automaton \( A \). By construction of the strategy, the goal mode must change when a play leads to one of these nodes. Under this assumption, it suffices to adjust how we reach a goal state, but once there the play will be at an existing node of \( A \).

In summary, the main steps of the proposed method are as follows. First compute a neighborhood \( \bar{N} \subseteq \Gamma \) of states based on proximity in the underlying continuous space, such that \( \bar{N} \) is sufficiently large to properly contain all states of nodes in \( \bigcup_i U_i \). See discussion in Section VI about possibilities for choosing \( \bar{N} \). For each goal mode \( i \) with nonempty \( U_i \),

1) Create the set \( N^i \) of nodes with goal mode \( i \) and corresponding state in \( \bar{N} \).
2) Create the set \( \text{Entry}^i \) of labels of nodes in \( N^i \) that can be entered from outside \( N^i \) in the original strategy. Also include in this set the current state of the play (assuming the algorithm is being used online, we must address how to move from the current position).
3) Compute the minimum \( \text{RA}_2(v) \) value (related to the reach annotation yielded by the automaton) over all nodes in \( \text{RA}^{-1}(\{i\} \times \text{Entry}^i) \cup U_i \), and call it \( m^* \).
4) Create the set \( \text{Exit}^i \) of states of nodes \( v \) in \( N^i \) with \( \text{RA}_2(v) \) less than \( m^* \). Precisely,

\[
\text{Exit}^i := \{ L(v) \mid v \in N^i; \text{RA}_2(v) < m^* \}.
\]

5) Compute \( \text{[Local}(\text{Exit}^i, \bar{N})] \).

- If \( \text{Entry}^i \) is not contained in \( \text{[Local}(\text{Exit}^i, \bar{N})] \), then the local problem is declared unrealizable and the neighborhood \( \bar{N} \) must be adjusted and the above steps repeated.
- Otherwise, compute the local strategy \( (A', \text{Exit}^i, \text{Exit}^i) \), and use it to mend the original strategy \( A \) by removing nodes in \( N^i \) and replacing it by the nodes in \( A' \), and adding appropriate edges corresponding to the edges into the entry states and the edges from the exit nodes in \( A \).

Intuitively, the sets are constructed such that if we can get to \( \text{Exit} \) from each node in \( \text{Entry} \), then from the definition of reach annotation it must be that distance from the current target system goal has not increased and that there is not a path under the new automaton \( A' \) leading back to \( \text{Entry} \).

To improve efficiency, \( \text{Pre} \) operations in the iterations are restricted to the neighborhood \( \bar{N} \). Note that upon reaching a “border node,” we must have arrived at a state in \( \text{Entry} \) (by definition) and therefore restricting the \( \text{Pre} \) operation to not go outside \( \bar{N} \) does not constrain the method.
When synthesizing for each $U_i$, incorporate the reach annotation from the patch into the original automaton $A$ by scaling all $L_3$ values of $A$ such that there is sufficient “room” for the new range of reach annotation values from the patch, after offsetting by the value at the lower attachment point. The resulting automaton $A'$ then has a labeling $L'_3$ that yields a reach annotation, as shown in Section III-D. Thus the proposed method can be applied to $A'$, the output of which can again be patched, and so on.

C. Formal statement

A precise statement appears in Algorithms 1 and 2. Some clarifications follow.

- $\bar{N} \subset \Gamma$ is a precondition asserting a set of neighborhood states is given. Note that $\bar{N}$ may be defined by a predicate on $\lambda \cup \gamma$.
- Line 20 finds the minimum positive integer that ensures the distance between the Entry and Exit sets (given by $m^* - m_s$) is greater than the range of reach annotation values in the local strategy (given by $m_{loc}$).

### Algorithm 1 Find local strategies

1: **INPUT:** GR(1) formula $\varphi$, strategy $A$, reach annotation $RA$, modified formula $\varphi'$, neighborhood $\bar{N} \subset \Gamma$
2: **OUTPUT:** set of triples $(A^i, Entry^i, Exit^i)$ with partial reach annotation $RA_i$
3: $\text{Patches} := \emptyset$
4: **for all** $i$ such that $U_i \neq \emptyset$ **do**
5: \hspace{0.5cm} $N^i := \{ v \in V \mid L(v) \in \bar{N}, RA(v) = i \}$
6: \hspace{0.5cm} **if** $U_i \setminus N^i \neq \emptyset$ **then**
7: \hspace{1.0cm} error --- $N^i$ is too small.
8: \hspace{0.5cm} **end if**
9: \hspace{0.5cm} Entry$^i$ := \{ $L(v)$ $|$ $v \in N^i$, $\exists u \in V \setminus N^i : u \rightarrow v$ \}
10: \hspace{0.5cm} $m^* := \min_{v \in U_i \cup \{ L^{-1} (Entry) \cap RA_2 (v) \}} RA_2 (v)$
11: \hspace{0.5cm} Exit$^i$ := \{ $L(v)$ $|$ $v \in N^i$, $RA_2 (v) < m^*$ \}
12: \hspace{0.5cm} $Z := \{ \text{Local}(Exit^i, N^i) \}$
13: \hspace{0.5cm} **if** Entry$^i \notin Z$ **then**
14: \hspace{1.0cm} error --- local problem unrealizable.
15: \hspace{0.5cm} **else**
16: \hspace{1.0cm} Synthesize $A^i$ such that $(A^i, Entry^i, Exit^i)$
17: \hspace{1.0cm} **with** partial reach annotation $RA_i$
18: \hspace{0.5cm} Patches := Patches $\cup$
19: \hspace{1.0cm} $\{ (A^i, Entry^i, Exit^i), N^i, RA_i \}$
20: **end if**
21: **end for**
22: **return** Patches

### Algorithm 2 Merge local strategies into the original

1: **INPUT:** GR(1) formula $\varphi$, strategy $A$, reach annotation $RA$, modified formula $\varphi'$, neighborhood $\bar{N} \subset \Gamma$, and Patches from Algorithm 1.
2: **OUTPUT:** Strategy automaton $A'$ for $\varphi'$ with reach annotation $RA'$
3: **for all** $(A^i = (V^i, \delta^i, L^i), Entry^i, Exit^i), N^i, RA_i)$ **in** Patches **do**
4: \hspace{0.5cm} **for all** $s \in Entry_i$ **do**
5: \hspace{1.0cm} find $u \in V^i$ such that $L(u) = s$
6: \hspace{1.0cm} **for all** $v \in V^i$ such that $v \rightarrow \delta u$ for some $u' \in N^i$ with $L(u') = s$ **do**
7: \hspace{1.5cm} replace transition $v \rightarrow u'$ of $\delta$ by $v \rightarrow s u$
8: \hspace{1.0cm} **end for**
9: \hspace{0.5cm} **end for**
10: \hspace{0.5cm} $m_s := \max_{v \in V^i} RA_2 (v)$
11: \hspace{0.5cm} **for all** $v \in V^i$ **do**
12: \hspace{1.0cm} $RA_2 (v) := RA_2 (v) + m_s$
13: **end for**
14: \hspace{0.5cm} $\alpha := \min_{k \in \mathbb{Z}_+, k(m_s - m_s) > m_{loc}} k$
15: \hspace{0.5cm} **for all** $v \in V^i$ with $RA_2 (v) = i$ **do**
16: \hspace{1.0cm} $RA_2 (v) := \alpha \cdot RA_2 (v)$
17: **end for**
18: **for all** $s \in Exit_i$ **do**
19: \hspace{0.5cm} $m_s := \max_{v \in V^i} RA_2 (v)$
20: \hspace{0.5cm} **for all** $v \in V^i$ **do**
21: \hspace{1.0cm} $RA_2 (v) := RA_2 (v) + m_s$
22: **end for**
23: \hspace{0.5cm} $A' := (V, \delta, L)$ and $RA' := RA$
24: **end for**
25: **end for**

D. Analysis

We begin with some remarks about the algorithm.

**Remark 6:** For each $i \in \{0, \ldots, n - 1\}$, Exit$^i$ (see definition on Line 11 of Algorithm 1) does not share any states with Entry$^i$ or $L(U_i)$. Precisely,

$Exit^i \cap (Entry^i \cup L(U_i)) = \emptyset$.

As described in Section II-A, the safety formulae $\rho_e$ and $\rho_s$ of a GR(1) specification can be equivalently expressed by a game graph $(\Gamma, E_e, E_s)$. In our analysis below we consider changes to $\rho_e$ and $\rho_s$, which is equivalent to the removal or addition of edges in $E$, denoted by $E \triangle E'$ (where $E'$ is the new edge set). The other parts of the specification remain unchanged.

Let $\varphi$ be a GR(1) specification as in equation (1). Denote the corresponding game graph by $\Gamma, E_e, E_s$. Suppose that the safety formulae $\rho_e$ and $\rho_s$ of $\varphi$ are changed, and call the modified specification $\varphi'$ with safety formulae $\rho'_e$ and $\rho'_s$. Let $A = (V, \delta, L)$ denote the original strategy automaton whose $L$ yields a reach annotation, and suppose $\bar{N} \subset \Gamma$ is chosen such that the patching problem is feasible, i.e., the algorithm terminates producing a modified automaton $A' = (V', \delta', L')$.

**Theorem 7:** The output of Algorithm 2, namely, $A'$ and $RA'$ such that $A'$ is a strategy automaton for $\varphi'$ and $RA'$ is a reach annotation on $A'$ with respect to $\varphi'$. Hence, $A'$ is a winning strategy automaton for $\varphi'$.
Proof: By Theorem 3, it suffices to show that \( A' \) is a strategy automaton, and that \( RA' \) is a reach annotation for it. For each system goal mode \( i \), on lines 24–29 of Algorithm 2 all nodes labeled with states affected by the GR(1) formula change to \( \varphi' \) are replaced. Thus, all nodes in \( A' \) have an outgoing edge for each possible environment move, leading to a permissible system move. Therefore \( A' \) is a strategy automaton.

By hypothesis, the original strategy automaton \( A \) has a reach annotation \( RA \) and each patch automaton \( A' \) such that \( (A', Entry', Exit') \) has partial reach annotation \( RA' \). If describing a strategy automaton, a partial reach annotation may be regarded as a reach annotation with constant \( RA_0' \). From these observations, it suffices to consider the transitions from \( A \) into \( A' \), and \( A' \) into \( A \). Lines 10–23 of Algorithm 2 ensure that \( RA_2 \) is nonincreasing across these transitions. It follows that \( RA' \) is a reach annotation on \( A' \).

IV. Example Scenarios and Experiments

The selection of the neighborhood \( \tilde{N} \) plays a crucial role in our proposed method, but may not admit a general form. Instead, one may need to have templates for constructing \( \tilde{N} \) for various problem classes and hybrid control systems. Below we consider discovery of a static obstacle online. Some other possibilities include the following.

- \( \tilde{N} \) could be a norm ball centered at some point of interest and scaled as needed. This is the heuristic used in [10], where the radius is incremented until all local problems are realizable, or the global problem is recovered.
- \( \tilde{N} \) could be obtained from the short-time reachable space computed for the given robot dynamics. This could be useful for under-actuated systems.
- If a nominal plan is obtained using a gradient-based method as in [14], then \( \tilde{N} \) may be iteratively expanded toward a goal, while allowing sufficient “width” for reactivity in an unknown environment.
- To find an appropriate neighborhood \( \tilde{N} \) after obtaining a finer resolution cell decomposition, include all strategy automaton nodes with state corresponding to refined cells. Depending on dynamics and continuity of the underlying system, we also include immediately neighboring cells in \( \tilde{N} \). Selecting which unrefined cells to include appears easier in some cases, such as for linear systems, and is the subject of future work.

A. Unreachable cells

One of the basic approaches to robotic navigation is decomposition of the workspace (e.g., an office floor) into finitely many cells (for an overview, see [15] and [6]). If we can assure the existence of dynamical control laws for steering the robot among these cells, then the cell decomposition may be representable as a finite directed graph, and hence planning over the workspace becomes a reactive synthesis problem. To cope with uncertainty of the space being navigated, a robot may build and update a map as new sensor data is collected online. An important class of such changes is the case where a cell becomes unnavigable. For instance, there may be a static obstacle. In this scenario, the strategy must be adjusted to move around the occupied cell. If the obstacle is small relative to the overall space addressed by the strategy, then we may expect the proposed method to be advantageous.

As preliminary evaluation of the proposed method, we conducted a simulation experiment for this scenario in a small gridworld problem. An illustration of the setting is given in Figure 1. Random gridworlds were generated for block densities of 0.1, 0.3, 0.5, and 0.7. E.g., a block density of 0.1 means that 10% of cells are occupied. At each time step, the robot can move “up,” “down,” “left,” or “right” (i.e., the world is 4-connected). There are two goals and one initial position, all randomly placed (but not overlapping). A single dynamic obstacle is also present. It must always eventually return to its base cell (labeled \( E \) in Figure 1), but on any particular instant may be at most one step away from its base (gray cells in Figure 1). The dynamic obstacles base cell is chosen randomly, and there may exist a plan for the robot to avoid it entirely. For each trial, a gridworld problem is randomly generated, solved to obtain a nominal strategy, and then a new static obstacle is introduced. The location of the new block is random but restricted to guarantee interference with the nominal strategy. For 0.7 density trials, the world has size \( 6 \times 20 \); for all others, it has size \( 4 \times 20 \).

The neighborhood \( \tilde{N} \) is formed by a \( 3 \times 3 \) subgraph centered on the newly blocked cell. Times required for global re-synthesis and times for our proposed method are shown in Figure 2, where two substantial outliers were excluded for clarity. Outliers were (319.25, 1.51) and (88.02, 2.18), where (global, patching) times are in seconds. Several statistics are given in Table II. Figure 3 shows mean run-times for global re-synthesis and the proposed method with respect to gridworld block density.

V. Experimental Demonstration

In addition to simulations, the patching algorithm was demonstrated on a physical setup consisting of a tracked
robot navigating through a planar environment with unexpected obstacles.

The environment through which the robot navigates is a $3\times3$ tiled floor populated with stationary obstacles. Each goal $J^s_i$ is inserted in software, and an automaton is synthesized to move between them. Patches to the strategy automaton are made as new obstacle position data become available online.

The robot used in these experiments was an iRobot LANdroid (Figure 4). The LANdroid is equipped with a Hokuyo scanning range finder (URG-04LX-UG01) and three IR LEDs, used in conjunction with an overhead position tracking system based on FView [16] to determine the position and heading of the LANdroid as it navigates through its environment. ROS [17] nodes were used to receive data from the Hokuyo and send control commands to the LANdroid, as well as to receive position information; the majority of processing was done off-robot. Plans and patches were synthesized using TuLiP\textsuperscript{1} and gr1c\textsuperscript{2}.

The discrete abstraction used in this hardware setup took the form of a quadtree data structure—refer to Figure 5 for an example. Nodes of the quadtree correspond to regions in a planar environment, and refinement and coarsification of quadtree nodes represent the splitting and joining of regions to and from four interior subregions. The level of refinement in a given quadtree cell corresponds to occupancy information retrieved from a probabilistic occupancy grid map populated by data from the laser range finder. The intermediate occupancy grid layer helps avoid unnecessary refinement of the quadtree due to noise.

The quadtree utilizes tesseral addressing [18] to allow for rapid adjacency calculations. Tesseral addresses are well-suited to representing hierarchical tessellations of planar spaces, as the quadtree in this implementation is doing. Here, tesseral addresses take the form of quaternary strings which uniquely address a region in a 2-D plane. For example, the tesseral address “30” would be read as “the 0th subregion of the 3rd subregion of the plane”. A labeled quadtree can be seen in Figure 5. In this implementation, the neighborhood $\hat{N}$ was determined based on cell adjacency and ancestry in the quadtree.

The process of patching in the context of these experiments can be seen in Figure 6. A video demonstration is available at http://vimeo.com/49653485. Initially, a path visiting two goals (green cells) is being pursued over the quadtree according to a nominal strategy automaton. As new occupancy data becomes available online, the automaton must be patched in response. $\hat{N}$ can be seen in red (the refined quadtree cell) and orange (non-refined neighbors); the partial resynthesis is made using cells within this neighborhood, and a correct strategy is recovered.
VI. CONCLUSION AND FUTURE WORK

It follows from Theorem 7 that given an appropriate initial strategy, we may patch-and-repeat indefinitely while maintaining correctness. However a critical issue in this process is selection of $\hat{N}$ and realizability of each successive specification. While these are informally explored in the computational experiments presented in Section IV, a thorough investigation remains for future work.

Current laboratory work described in Section V has focused on demonstrating viability and establishing a hardware setup to be used for future experiments. Future work will include statistical comparison of time required when patching and when fully resynthesizing. Additionally we plan to include nondeterministic elements in the robots environment, such as a moving obstacle.

The key insight of the present work is twofold. Let $A$ be the original strategy automaton. First, by finding the set $U_i$ of nodes in $A$ that are affected for each goal $J_i^*$, we exploit the case of problems where system goals correspond to disconnected regions of a continuous space. In this case, only efforts to reach some of the goals may be affected by a game graph change. For instance, this naturally occurs in robot surveillance problems, where different rooms in a building must be visited. Second, our local $\mu$-calculus formula (III-B) has alternation depth of one (cf. [19]), and indeed, given environment goals, is just a reachability computation around a cluster of affected vertices in the game graph. In future work we will investigate the computational complexity of the proposed method and conduct empirical studies in application to robot navigation.

ACKNOWLEDGMENTS

This work is partially supported by the Boeing Corporation, the Caltech Center for the Mathematics of Information (CMI), and a United Technologies Research Center postdoctoral fellowship.

REFERENCES