Submitted, 2012 International Conference on Robotics and Automation (ICRA)
http://www.cds.caltech.edu/~murray/papers/lmb12-icra.html

Backtracking temporal logic synthesis for uncertain environments

Scott C. Livingston, Richard M. Murray, and Joel W. Burdick

***Abstract*— This paper considers the problem of synthesizing correct-by-construction robotic controllers in environments with uncertain but fixed structure. "Environment" has two notions in this work: a map or "world" in which some controlled agent must operate and navigate (i.e. evolve in a configuration space with obstacles); and an adversarial player that selects continuous and discrete variables to try to make the agent fail (as in a game). Both the robot and the environment are subjected to behavioral specifications expressed as an assume-guarantee linear temporal logic (LTL) formula. We then consider how to efficiently modify the synthesized controller when the robot encounters unexpected changes in its environment. The crucial insight is that a portion of this problem takes place in a metric space, which provides a notion of nearness. Thus if a nominal plan fails, we need not resynthesize it entirely, but instead can "patch" it locally. We present an algorithm for doing this, prove soundness, and demonstrate it on an example gridworld.**

I. Motivation and introduction

Consider the hypothetical office floor map shown in Fig. 1, and suppose a robot is tasked with always visiting the lounge and printing room. If a special "espresso ready" flag is received, then the robot must eventually go to the coffee nook. Additionally, there are safety requirements, e.g.

- avoid collisions with *a priori* known walls (appearing in given map); and
- avoid collisions with short-term static (e.g. trash bins) and dynamic (e.g. people) objects in the area.

Given a floor map and assumptions about

- arrival rate of the "espresso ready" flag; and
- how static and dynamic obstacles can appear or move,

a controller M, generally taking the form of a hybrid system, can be synthesized to guarantee this behavior. During online operation, errors in the nominal map may be discovered. In this illustration, the "lounge" room has been newly partitioned with cubicles. Thus the map needs mending, as may the robot's control system.

Assuming the nominal controller M is large and computationally difficult to completely re-synthesize on the new map, the algorithm proposed in this paper can locally "patch" it to recover global correctness. It does so by exploiting the fact that most behavior in the office continues to be correct, if only we can negotiate the discovered change locally. Of course, the change may indeed have global consequences, and in that case, the entire control system will be resynthesized.

The setting and conclusions are more general than the repetitive navigation problem of this motivating example.

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

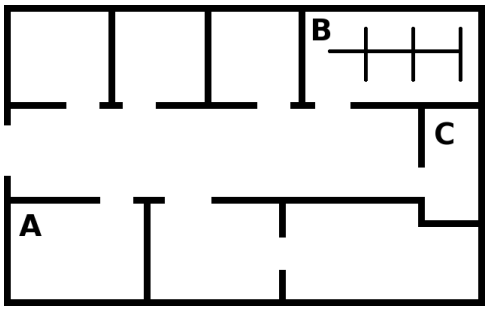

Fig. 1. Illustration of office space in the example. Room A is the "printing room," room B is the "lounge," and room C is the "coffee nook."

This paper considers the problem of "reactive synthesis," whose goal is to automatically create an autonomous machine that satisfies some given description of desired output behavior, no matter what input is applied, subject to constraints expressed in a temporal logic over infinite strings [1]; such a general statement goes at least back to Church [2].

In the hybrid control systems literature, two main approaches have been used to attack the reactive synthesis problem. The first approach uses methods from the computer aided verification literature for synthesizing the discrete portion of the reactive program. To use this approach, the underlying continuous dynamics of the robot and environment must already be abstracted into a discrete dynamical system, for which these tools are directly applicable [3], [4]. The discrete abstraction step must preserve, in a simulation sense [5], the connectivity and reachability of the abstracted regions in the underlying continuous space.

The second approach is to algorithmically find controllers that satisfy a temporal logic specification, or what has also been called symbolic motion planning (for an informal overview, see [6]). Several correct-by-construction methodologies have been considered. Kloetzer and Belta show how to find controllers for affine systems and propositions defined over partitions of state space [7]. Karaman and Frazzoli propose a solution found by sampling sequences of control inputs to a nonlinear system. To that end they propose an extension of RRT that allows cyclic behavior, called RRG [8].

Only recently has uncertainty been treated directly in work applying formal methods to robotics. Wongpiromsarn *et al.* treat uncertainty as disturbances acting on the underlying continuous dynamics [9]. Johnson and Kress-Gazit studied the effect of probabilistic errors in determining proposition values (motivated by sensor errors), providing some insight

into fragility of the original specification [10]. Belta *et al.* have framed uncertainty in the language of Markov decision processes and proposed algorithms for discrete policies that maximize probability of satisfying a given temporal logic specification [11], [12].

This paper considers, in the formal system framework, a type of uncertainty which has heretofore been largely ignored—potential topological changes in the robot's environment. To address this type of uncertainty, we try to bridge the gap of abstraction that previously formed a strong separation between the underlying continuous dynamics of a physical system and the discrete variables over which reactive synthesis occurs. That is, we wish to recover some of the structure of the underlying Banach space and use it to repair flawed models originally used to create nominal plans. Our approach is dubbed "backtracking reactive synthesis." Philosophically, our method can be considered a generalization of the concepts behind Stentz's D* algorithm [13], to the problem of creating correct and always running hybrid controllers. Whereas D* "patches" a motion plan in order to handle unforeseen problems in the environment, this paper "patches" a general hybrid automaton. The most similar prior work is that of Wongpiromsarn *et al.* [9], who show a method for decomposing a large reactive synthesis problem into a partial order of receding horizon subproblems. A major limitation in that work is the need to construct the decomposition by hand, i.e. no automatic horizon selection procedure is given. In this paper, we use the metric of the underlying space to provide a notion of "localness", which permits "patching" a nominal solution. The "patches" can be thought of as "short horizon problems," in the terminology of [9].

II. Definitions and problem setting

Let $\mathcal{Y}$ be a set of "system" variables, and $\mathcal{X}$ a set of "environment" variables, both finite and boolean. Throughout the paper these sets of variables are assumed disjoint, i.e. $\mathcal{X} \cap \mathcal{Y} = \varnothing$. Individual variables are written as uppercase letters, e.g. $V \in \mathcal{X}$. Write the set of valuations of the controlled variables as $\Sigma_{\mathcal{Y}}$; i.e. any particular setting of these variables is some $y \in \Sigma_{\mathcal{Y}}$. Note that what we call "valuations" is also variously called "states" (e.g., see [14]), but we reserve the word for continuous dynamics in a "state" space. The variables in $\mathcal{Y}$ may be labeled and ordered such that for $y \in \Sigma_{\mathcal{Y}}$, y^i denotes the valuation of the i-th variable in $\mathcal{Y}$ at y. Similar notation is used for the environment variables $\mathcal{X}$.

Let $\mathrm{form}_{\mathcal{Y}} : \Sigma_{\mathcal{Y}} \to \mathrm{Bool}(\mathcal{Y})$ be the map taking valuations of $\mathcal{Y}$ to boolean formulae such that the resulting formula is true precisely when the variables have the given valuation.

$$\mathrm{form}_{\mathcal{Y}}(y) \triangleq \left(\bigwedge_{i \in I_{\mathrm{T}}} y^i \right) \wedge \left(\bigwedge_{j \in I_{\mathrm{F}}} \neg y^j \right), \tag{1}$$

where the index sets I_{T} and I_{F} denote indices for which the variables are true and false, respectively. The domain can be restricted to a particular set of variables in the natural way; this is indicated by changing the subscript, e.g. $\mathrm{form}_{\mathcal{X}}$. When we are concerned with both sets of variables, we will drop the subscript and write

$$\mathrm{form} : \Sigma_{\mathcal{X}} \times \Sigma_{\mathcal{Y}} \to \mathrm{Bool}(\mathcal{X} \cup \mathcal{Y}),$$
$$\mathrm{form}(xy) = \mathrm{form}(x) \wedge \mathrm{form}(y).$$

An *execution trace* is a sequence of environment and system valuations, occurring as turns in a game; write such a sequence as

$$\sigma := x_0 y_0 x_1 y_1 \cdots,$$

where the subscripts indicate (discrete) time steps. For reactive synthesis, we wish to generate a strategy function

$$M : (\Sigma_{\mathcal{X}} \Sigma_{\mathcal{Y}})^* \Sigma_{\mathcal{X}} \to \Sigma_{\mathcal{Y}}$$

to select values for controlled variables that satisfy a behavioral specification, given the environment variables satisfy an assumption. Given the specification φ, an LTL formula over variables $\mathcal{X} \cup \mathcal{Y}$, we seek an M such that for any resulting trace σ, $\sigma \models \varphi$. Informally, this notation means "the execution σ is correct with respect to the specification φ."

TABLE I
Linear temporal logic (LTL) operators

$\vee$	"or"
$\wedge$	"and"
$\neg$	"not"
$\bigcirc$	"next"
$\Diamond$	"eventually"
$\Box$	"always" (safety)
$\Box\Diamond$	"infinitely often" (progress)
$\models$	"satisfies"

In this paper, we consider a restricted class of linear temporal logic (LTL) formulae with which the synthesis problem can be efficiently solved, called GR(1) (Generalized Reactivity) [15], [16]. A GR(1) formula φ has the form

$$\begin{aligned} \psi_e &:= \left(\bigwedge_{i \in I_{e,t}} \Box \psi_{e,t_i} \right) \wedge \left(\bigwedge_{i \in I_{e,g}} \Box\Diamond \psi_{e,g_i} \right) \\ \psi_s &:= \left(\bigwedge_{i \in I_{s,t}} \Box \psi_{s,t_i} \right) \wedge \left(\bigwedge_{i \in I_{s,g}} \Box\Diamond \psi_{s,g_i} \right) \\ \varphi &:= (\psi_e \wedge \psi_{\mathrm{init}}) \implies \psi_s, \end{aligned} \tag{2}$$

where ψ_{e,t_i} are state formulae defined over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X}$, ψ_{s,t_i} are state formulae defined over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc(\mathcal{X} \cup \mathcal{Y})$, ψ_{e,g_i} are state formulae defined over $\mathcal{X} \cup \mathcal{Y}$, and ψ_{init} is a state formula describing initial conditions on $\mathcal{X} \cup \mathcal{Y}$. We refer to subformulae of the form $\Box\psi_{e,t_i}$ as "safety" properties, and subformulae of the form $\Box\Diamond\psi_{e,g_i}$ as "progress" properties. The index sets for the subformulae are $I_{e,t}$, $I_{e,g}$, $I_{s,t}$, $I_{s,g}$.

There are two important aspects of GR(1) formulae. First, the specification has an assumption-guarantee structure. Second, only two types of properties are specified, and they affect the problem in distinct ways. Given current values,

safety properties restrict what values the variables can take next. Hence "safety" provides the machinery for expressing reachability for a discrete abstraction of an underlying continuous system. Progress properties describe behaviors that occur "infinitely often". A progress formula may not be true for a particular step but will be true in finite time. E.g., this can be used to describe surveillance behavior by a robot. For related model checking theory, see [17], [18].

A solution strategy M is represented by a finite automaton that takes environment valuations (actions) as input.

Definition 1: A *finite automaton* is a tuple

$$M = (\Sigma_{\mathcal{X}}, S, S_0, g, \delta),$$

where the set of environment valuations $\Sigma_{\mathcal{X}}$ is the input alphabet, S is a finite set of nodes, $S_0 \subseteq S$ initial nodes, $\delta : S \times \Sigma_{\mathcal{X}} \to S$ is the transition function, and $g : S \to \Sigma_{\mathcal{Y}}$ labels nodes with system output.

Without loss of generality and since we do *not* require node labels to be unique, we assume throughout that the transition function δ is injective with respect to $\Sigma_{\mathcal{X}}$, i.e.

$$x_1 \neq x_2 \implies \delta(v, x_1) \neq \delta(v, x_2) \qquad \text{for all } v \in S.$$

Definition 2: The set of *edges* of M is

$$\mathrm{E}(M) \triangleq \{(v, v') \in S \times S \mid \exists x \in \Sigma_{\mathcal{X}} \ . \ \delta(v, x) = v'\}.$$

With this edge set, we may regard M as a directed graph. From the assumed injectivity of δ with respect to $\Sigma_{\mathcal{X}}$, an *edge labeling function* is immediate

$$h : E(M) \to \Sigma_{\mathcal{X}},$$
$$h((v, v')) = x \quad \text{iff} \quad \delta(v, x) = v'.$$

The sets of edges going into and out of a node $v \in S$ are

$$\begin{aligned} \mathrm{In}(v) &= \{(v', v) \in E(M) \mid v' \in S\} \\ \mathrm{Out}(v) &= \{(v, v') \in E(M) \mid v \in S\}. \end{aligned}$$

An execution trace $\sigma = x_0 y_0 x_1 y_1 \ldots$ leads to an *execution path* $\rho = s_0 s_1 s_2 \ldots$ of the nodes $s_i \in S$ visited by M given the input string $x_0 x_1 \ldots$. From the labeling g, an execution path yields a sequence of system outputs (or actions) $y_0 y_1 \ldots$.

The patched strategies generated by the algorithm in Section III have finite memory. Nodes are able to clear or set memory, and transitions may be conditioned on memory contents. Since the memory is finite, such "finite memory automata" are equivalently expressive to those defined above. The extension is only a matter of notation, and simplifies our treatment.

Let $Z \subseteq \mathbb{R}^n$ open, and $U \subseteq \mathbb{R}^m$ be a set of control inputs, and let the dynamics be

$$z_{k+1} = f(z_k, u_k), \tag{3}$$

where f is smooth. We call Z the "continuous state space" and sometimes Eq. (3) the "underlying continuous system." A cellular decomposition $\mathcal{P}$ of Z is chosen such that it is proposition-preserving and satisfies a reachability condition (defined below; also see [5]).

For any cell $\zeta_j \in Z/\mathcal{P}$, a valuation $y \in \Sigma_{\mathcal{Y}}$ is said to take place in that cell if it implies that the continuous state z is in ζ_j. In this case, we write

$$y \implies \zeta_j.$$

Definition 3: Let $\zeta_i, \zeta_j \in Z/\mathcal{P}$. ζ_j is said to be *reachable* from ζ_i if for any $z_0 \in \zeta_i$, there exists some finite control sequence $\mu : \{0, \ldots k-1\} \to U$ such that applying μ to dynamics Eq. (3) with initial condition z_0 yields a final state of $z_k \in \zeta_j$.

Definition 4: $V \in \mathcal{X} \cup \mathcal{Y}$ is said to be a *spatially-dependent variable* if its truth-value is a function of the continuous state Z, written $V : Z \to \{\mathrm{True}, \mathrm{False}\}$.

Definition 5: Let $S' \subseteq S$, and let $\{\zeta_k\}_k \subseteq Z/\mathcal{P}$, where $k \in K$ is an index set. We define the *region nodes* to be

$$\mathrm{Reg}(S', \{\zeta_k\}_k) \triangleq \{v \in S' \mid \exists k \in K \ . \ g(v) \implies \zeta_k\}.$$

Intuitively, $\mathrm{Reg}()$ is the set of nodes that satisfy some set of cells of the continuous space — i.e., if the valuation of the node holds (or equivalently, if an execution path has led to that node), then the underlying continuous system must be in one of the given cells.

Definition 6: Let $\gamma > 0$. The *abstract neighborhood* is

$$B_\gamma(\zeta_i) \triangleq \{\zeta_j \in Z/\mathcal{P} \mid \exists z_1 \in \zeta_i, z_2 \in \zeta_j \ . \ \|z_1 - z_2\| < \gamma\}.$$

For completeness, define $B_0(\zeta_i) = \{\zeta_i\}$.

Intuitively, the abstract neighborhood lifts the ball of radius γ from the continuous space up into the discrete abstraction in such a way that mapping the B_γ back down into the continuous space will cover the original norm ball.

Definition 7: Let $S' \subseteq S$, and let ψ be a boolean formula. We define the *satisfying nodes* to be

$$\begin{aligned} \mathrm{Sat}(S', \psi) \triangleq \Big\{ & v' \in S' \mid \exists e' \in \mathrm{In}(v') \ . \\ & h(e') g(v') \models \psi \Big\} \end{aligned}$$

Intuitively, $\mathrm{Sat}()$ gives all nodes for which the corresponding valuation label and some inward edge label satisfies the given formula.

Definition 8: Let M be a solution automaton, and let $S' \subseteq S$. Then the set of *exit nodes* is defined

$$\begin{aligned} \mathrm{Exit}(S') \triangleq \Big\{ & v_1 \in S' \mid \exists x \in \Sigma_{\mathcal{X}}, v_2 \in S \setminus S' \ . \\ & \delta(v_1, x) = v_2 \Big\}. \end{aligned}$$

Or equivalently,

$$\mathrm{Exit}(S') \triangleq \{v_1 \in S' \mid \exists v_2 \in S \setminus S' \ . \ (v_1, v_2) \in E(M)\}.$$

The set of *entry nodes* is defined in a similar manner,

$$\begin{aligned} \mathrm{Entry}(S') \triangleq \Big\{ & v_1 \in S' \mid \exists x \in \Sigma_{\mathcal{X}}, v_2 \in S \setminus S' \ . \\ & \delta(v_2, x) = v_1 \Big\}. \end{aligned}$$

Definition 9: Let $l \in S'$, where $S' \subseteq S$. Define the *restricted reachable set* to be

$$\begin{aligned} \mathrm{Reach}^*(l, S') \triangleq \Big\{ & v \in S' \mid \langle l, p_1, \ldots, p_k, v \rangle \text{ is a path with} \\ & p_1, \ldots, p_k \in S' \Big\} \end{aligned}$$

The path of zero length is included, i.e. $l \in \mathrm{Reach}^*(l, S')$.

III. Algorithm

A. Overview

We assume there is a nominal robot controller M that realizes a (global) specification φ. This controller is "nominal" in that it was constructed based on a model of the world $Z/\mathcal{P}$, wherein the robot can move among cells of interest subject to some dynamics. If from online sensing one of the cells $\zeta_i \in Z/\mathcal{P}$ is discovered to be unreachable, then M must be corrected if the robot is still to satisfy φ. Hence initialization of our algorithm.

Execution begins when a cell $\zeta_i \in Z/\mathcal{P}$ expected to be reachable turns out not to be. Suppose this is first sensed when the automaton M is at node $v \in S$. There are two main parts to our approach:

1) find a neighborhood around ζ_i such that a set of local specifications $\{\varphi_l\}_l$ can be realized; and
2) replace defunct parts of M with the synthesized local patches $\{M_{\text{new},l}\}_l$.

1) Finding locally realizable specifications: The first step proceeds by iteratively considering larger neighborhoods until local patches are realized or the global problem is obtained. The latter eventually occurs because for a sufficiently large neighborhood, synthesizing patches is as hard as resynthesis of the entire strategy M. If the global problem has become infeasible, then terminate with failure.

The sets of spatially-dependent environment and system variables are $\mathcal{X}_{\text{spatial}}$ and $\mathcal{Y}_{\text{spatial}}$, respectively. By the assumption that the initial cellular decomposition $\mathcal{P}$ of the state space Z is proposition-preserving, the notion of neighborhood for our purposes is some subset of $Z/\mathcal{P}$. Hence the definition of abstract neighborhood, $B_{r_{\text{inc}}}(\zeta_i) \subseteq Z/\mathcal{P}$. The radius r_{inc} is iteratively incremented by γ, which has the same units as the continuous state space Z. Given that the control strategy will change only when the robot is in this neighborhood, some of the spatially-dependent variables will have fixed truth-value and thus need not be included in the patch synthesis problem. The restricted set of variables is called $\mathcal{X}_{\text{inc}} \cup \mathcal{Y}_{\text{inc}}$ in the algorithm.

Since we seek to correct "local" behavior, all parts of the original strategy M that correspond to the robot being in the neighborhood $B_{r_{\text{inc}}}(\zeta_i) \subseteq Z/\mathcal{P}$ must be replaced. The corresponding set of nodes in M is $\text{Reg}(S, B_{r_{\text{inc}}}(\zeta_i))$. This set is important for two purposes. First, any system goals ψ_{s,g_i} that are satisfied at some node in it must be included as local goals in the patch. Otherwise the environment might be able to drive the system into this patched cell indefinitely, leading to a specification violation.

Second, by examining how nodes from $S' := \text{Reg}(S, B_{r_{\text{inc}}}(\zeta_i))$ connect with the remainder of the strategy automaton, entry and exit points for patches can be determined. Specifically, the entry nodes are $\text{Entry}(S') \cup \{v\} \cup \text{Init}$. Nodes in $\text{Entry}(S')$ are just those from the original M that have an incoming transition from a node outside the abstract neighborhood. Thus for an execution path to lead from outside a patch to $l \in \text{Entry}(S')$, the continuous state of the robot must transition from outside to in $B_{r_{\text{inc}}}(\zeta_i)$.

The other set of entry nodes, $\{v\} \cup \text{Init}$, handles special cases. v is the node at which the unreachable cell ζ_i was discovered. We must ensure initialization from v is satisfiable since we want the algorithm to perform patching online—with no need for "restarts." Nodes in Init are those from which the robot can be started in the neighborhood $B_{r_{\text{inc}}}(\zeta_i)$, thus possibly never encountering a transition into it.

Complementary to entry nodes l, we must find exit points to return to M after local correction $M_{\text{patch},l}$, except in a special case described below. To ensure correctness of the output of the algorithm M_{new} with respect to the global specification, a list of possible exit points is formed of those nodes that

- were reachable in M from l by paths restricted to S', and
- correspond to continuous states z that can be driven out of the neighborhood $B_{r_{\text{inc}}}(\zeta_i)$ in one (abstract) transition.

This set is $\text{Reach}^*(l, S') \cap \text{Exit}(S')$.

As noted earlier, the local patch may be required to satisfy some global system goals. Therefore it does not suffice to immediately return to M when execution in a patch automaton reaches one of the above exit nodes. In the special case that these system goals were only satisfied in the abstract neighborhood, the environment might drive the system to the same exit node every time a patch is visited and prevent infinite-often satisfaction of these goals. This case is addressed by adding finite memory $\{\xi_i\}_i$ to the strategy. The goal indicators $\{\xi_i\}_i$ allow an execution path that enters the patch to remain there until all local system goals are satisfied. Of course, such "indicators" could be incorporated into the system variables $\mathcal{Y}$ without loss of generality.

There is a special case where, for some entry node l,

$$\text{Reach}^*(l, S') \cap \text{Exit}(S') = \varnothing.$$

If such a node l was reached by M during an execution trace, then the continuous state z would enter the neighborhood and never leave it. Since the given controller M is assumed to be correct for the nominal model, such an execution trace is in fact correct (nominally). It follows that all global system goals could be satisfied using M restricted to the abstract neighborhood, hence a patch $M_{\text{patch},l}$ beginning at l would not need to eventually return to M.

From the above description, a patch synthesis problem with an appropriate "patch" specification φ_l is rooted at each entry node. If all $\{\varphi_l\}_l$ are realizable, then the algorithm has succeeded. It remains only to merge the results $\{M_{\text{patch},l}\}_l$ into the original M.

2) Patching-in the corrections: There are two major steps to merge each patch automaton $M_{\text{patch},l}$ into M. First, recall that l is an "entry" node in the original M. By definition of φ_l, the truth-value of environment and system variables at l is matched by some initialization node in $M_{\text{patch},l}$; call it l'. Each outgoing transition from l that matches (i.e. has

same environment label x) an outgoing transition from l' is replaced such that the new transition leads into $M_{\mathrm{patch},l}$. Thus the patch automaton is "entered" when the continuous state z of the robot enters the neighborhood $B_{r_{\mathrm{inc}}}(\zeta_i)$.

Second, if there is no exit point (recall the special case noted earlier), then remaining in $M_{\mathrm{patch},l}$ is correct. Otherwise, there must be some way to leave $M_{\mathrm{patch},l}$. By definition of φ_l, the label of at least one "exit" node $p \in \mathrm{Reach}^*(l, S') \cap \mathrm{Exit}(S')$ in M must be satisfied infinitely often for an execution contained entirely in $M_{\mathrm{patch},l}$. Thus after entering $M_{\mathrm{patch},l}$, an execution path will always eventually be able to return to M at a transition reachable before patching. Now using the local goal indicators $\{\xi_i\}_i$, exiting $M_{\mathrm{patch},l}$ can only occur after all locally satisfiable goals are met. Because they are guaranteed to be met eventually by φ_l, matching nodes in $M_{\mathrm{patch},l}$ can have outward transitions augmented to allow returning to M when all indicators $\{\xi_i\}_i$ are True.

Once all patches $\{M_{\mathrm{patch},l}\}_l$ are merged as above, the invalid nodes $\mathrm{Reg}(S, \{\zeta_i\})$ corresponding to reaching the unreachable ζ_i can be removed. Return the final solution M_{new}.

B. Formal statement

The algorithm is listed in Figs. 2 through 10. The division into two blocks of code follows the "two part" description given in the overview of the previous section.

Clarifications of code in Fig. 2:

- Line 4 checks whether the abstract neighborhood is sufficiently large to warrant global re-synthesis.
- Lines 8–15 build sets of environment and system variables that could change in the local problem—in particular (line 11), those spatially-dependent variables that could change value for some continuous trajectory in the abstract neighborhood.
- Line 23 defines initial conditions for the patch specification. This ensures correct "entry points."
- Line 24 accounts for the special case that $I'_{s,g} = I_{s,g}$ and from entry point l an execution path will never leave the neighborhood. Hence local navigation goal $\psi_{\mathrm{patch},l}$ is set to be vacuously satisfied on line 25.
- Lines 29–31 define exit conditions for the patch specification. These ensure possible "exit points."
- Lines 32–34 include safety properties and global goals that are relevant in the local problem. Line 34 defines the patch specification φ_l.

Clarifications of code in Fig. 3:

- Line 1 creates a new node set by taking the disjoint union of the original node set and those of all patch strategies. In the pseudocode, references to $S_{\mathrm{patch},l}$, $\mathrm{Reg}(S, \{\zeta_i\})$, etc. are to the corresponding subsets of S_{new}.
- Lines 7–14 find all nodes from $M_{\mathrm{patch},l}$ that share labeling with l. Edges are then replaced to cause M_{new} to transition into $S_{\mathrm{patch},l}$ when appropriate.
- Line 15 finds all nodes in $S_{\mathrm{patch},l}$ that satisfy one of the disjuncts in $\psi_{\mathrm{patch},l}$ from the patch specification φ_l.
- Lines 16–23 find among the possible exit nodes in the original strategy M those which match one of the patch exit nodes in S_{exit}. Outgoing transitions are appended to $p' \in S_{\mathrm{new},l}$. Conditions are added such that a transition out of the patch can only be taken if all local goals have been met at least once (Line 20). Otherwise, execution must continue in the patch (Line 21).
- Lines 24–30 add memory rules to nodes such that appropriate indicators are set when system goals are satisfied.

IV. Analysis

Several sets are invoked in the algorithm in such a way that, if they were empty, then the algorithm would fail to run. We show this is not the case. Let ζ_i be the unreachable cell, let $S' := \mathrm{Reg}(S, B_{r_{\mathrm{inc}}}(\zeta_i))$, and let $l \in \mathrm{Entry}(S') \cup \{v\} \cup \mathrm{Init}$.

Remark 1: Merging only occurs at nodes corresponding to reachable cells, i.e.

$$\mathrm{Entry}(S') \cap \mathrm{Reg}(S, \{\zeta_i\}) = \mathrm{Exit}(S') \cap \mathrm{Reg}(S, \{\zeta_i\}) = \varnothing$$

Remark 2: Q in line 7 of Fig. 3 is nonempty. Nonemptiness follows from realizability of φ_l —in particular, its initial conditions $\psi'_{\mathrm{init},l}$.

Remark 3: S_{exit} is nonempty. $M_{\mathrm{patch},l}$ is a realization of φ_l, thus by construction of the patch goals $\psi_{\mathrm{patch},l}$, at least one match will be found.

As stated, *the algorithm is not complete* due to the simplifying assumption regarding environment behavior in the patch cell. We do not require the adversarial environment to satisfy all of its global "infinitely often" behaviors. A simple counterexample is navigating a vehicle through a thin passageway. Under the patch assumption, it could be blocked indefinitely by an environmentally-controlled obstacle. In contrast, the global specification could require it to clear the passageway always eventually, hence realizability.

However, increasing the radius r_{inc} causes the algorithm to reduce to the global problem after a finite number of iterations. So in that (degenerate) sense, it is complete.

Theorem 1: Any successfully patched controller M_{new} realizes the original specification Eq. (2).

Proof: Let ζ_i be the cell discovered to be unreachable. Let $N = B_{r_{\mathrm{inc}}}(\zeta_i) \subseteq Z/\mathcal{P}$ be the abstract neighborhood. Set r_{inc} to its value at algorithm termination, and let $S' = \mathrm{Reg}(S, B_{r_{\mathrm{inc}}}(\zeta_i))$.

Aside from initial conditions, the assumption of the global specification Eq. (2) implies the assumptions of all patch specifications $\{\varphi_l\}_l$ because the latter is a proper subset of the former. We can ignore the initial conditions $\psi'_{\mathrm{init},l}$ of the patch specification since by construction they are reachable in the original automaton M from its own initial conditions ψ_{init}.

There are two cases to consider. First, for any execution path that is a subset of $S \setminus \mathrm{Reg}(S, \{\zeta_i\})$, the global specification is satisfied by hypothesis.

It remains to address execution paths that include nodes from $S_{\mathrm{patch},l}$ for some l. There are two basic possibilities

Fig. 2. Main loop

Require: $\zeta_i \in Z/\mathcal{P}$ unreachable, v node in M

1: $r_{\text{inc}} \leftarrow 0$
2: **repeat**
3: $r_{\text{inc}} \leftarrow r_{\text{inc}} + \gamma$
4: **if** $Z/\mathcal{P} = B_{r_{\text{inc}}}(\zeta_i)$ **then**
5: **abort** //recovered global problem
6: **end if**
7: $S' \leftarrow \text{Reg}(S, B_{r_{\text{inc}}}(\zeta_i))$
8: **for** $\mathcal{V}$ **in** $\{\mathcal{X}, \mathcal{Y}\}$ **do**
9: $\mathcal{V}_{\text{inc}} \leftarrow \mathcal{V} \setminus \mathcal{V}_{\text{spatial}}$
10: **for** V **in** $\mathcal{V}_{\text{spatial}}$ **do**
11: **if** $V(B_{r_{\text{inc}}}(\zeta_i)) = \{\text{True}, \text{False}\}$ **then**
12: $\mathcal{V}_{\text{inc}} \leftarrow \mathcal{V}_{\text{inc}} \cup \{V\}$
13: **end if**
14: **end for**
15: **end for**
16: **for** α **in** $\{e, s\}$ **do**
17: $I'_{\alpha,t} \leftarrow \texttt{FindTrans}(\mathcal{X}_{\text{inc}}, \mathcal{Y}_{\text{inc}}, \alpha)$
18: $I'_{\alpha,g} \leftarrow \texttt{LocalGoals}(B_{r_{\text{inc}}}(\zeta_i), \alpha)$
19: **end for**
20: $\text{Init} \leftarrow S_0 \cap \text{Reg}(S, B_{r_{\text{inc}}}(\zeta_i))$
21: $\text{Soln} \leftarrow \varnothing$ //set of patch automata
22: **for** l **in** $\text{Entry}(S') \cup \{v\} \cup \text{Init}$ **do**
23: $\psi'_{\text{init},l} \leftarrow \bigvee_{e \in \text{In}(l)} \text{form}(h(e)g(l))$
24: **if** $\text{Reach}^*(l, S') \cap \text{Exit}(S') = \varnothing$ **then**
25: $\psi_{\text{patch},l} \leftarrow \text{True}$ //special case
26: **else**
27: $\psi_{\text{patch},l} \leftarrow \text{False}$
28: **end if**
29: **for** $p \in \text{Reach}^*(l, S') \cap \text{Exit}(S')$ **do**
30: $\psi_{\text{patch},l} \leftarrow \psi_{\text{patch},l} \vee \left(\bigvee_{e \in \text{In}(p)} \text{form}(h(e)g(p))\right)$
31: **end for**
32: $\psi'_e \leftarrow \left(\bigwedge_{i \in I'_{e,t}} \Box\psi_{e,t_i}\right) \wedge \left(\bigwedge_{i \in I'_{e,g}} \Box\Diamond\psi_{e,g_i}\right)$
33: $\psi'_s \leftarrow \left(\bigwedge_{i \in I'_{s,t}} \Box\psi_{s,t_i}\right) \wedge \left(\bigwedge_{i \in I'_{s,g}} \Box\Diamond\psi_{s,g_i}\right)$
34: $\varphi_l \leftarrow \psi'_{\text{init},l} \wedge \psi'_e \implies \psi'_s \wedge \Box\Diamond\psi_{\text{patch},l}$
35: **if** φ_l realizable **then**
36: $M_{\text{patch},l} \leftarrow \texttt{Synthesize}(\mathcal{X}_{\text{inc}}, \mathcal{Y}_{\text{inc}}, \varphi_l)$
37: $\text{Soln} \leftarrow \text{Soln} \cup \{(l, M_{\text{patch},l}, \varphi_l)\}$
38: **else**
39: **goto line** 3 //failed; try next larger radius
40: **end if**
41: **end for**
42: **return** Soln
43: **until** all φ_l realizable

Fig. 3. Create new strategy M_{new}

Require: $\text{Soln} = \{(l, M_{\text{patch},l}, \varphi_l)\}_l$

1: $S_{\text{new}} \leftarrow S \uplus (\biguplus_l S_{\text{patch},l})$
2: $\text{Mem} \leftarrow \{\xi_i\}_{i \in I'_{s,g}}$ //goal indicator memory
3: $\text{Rule} \leftarrow \texttt{RuleClear}$ //nodes clear memory by default
4: $\text{Cond} \leftarrow \texttt{NULL}$ //no conditional transitions by default
5: $M_{\text{new}} \leftarrow (\Sigma_{\mathcal{X}}, S_{\text{new}}, S_0, g, \delta, \text{Mem}, \text{Rule}, \text{Cond})$
6: **for** $(l, M_{\text{patch},l}, \varphi_l)$ **in** Soln **do**
7: $Q \leftarrow \texttt{FindMatches}(l, S_{\text{patch},l})$
8: **for** (l, w) **in** $\text{Out}(l)$ **do**
9: **for** $((l, w), l', (l', w'))$ **in** Q **do**
10: $E(M_{\text{new}}) \leftarrow E(M_{\text{new}}) \cup (l, w')$
11: **delete** (l, w) **from** $E(M_{\text{new}})$
12: **delete** (l', w') **from** $E(M_{\text{new}})$
13: **end for**
14: **end for**
15: $S_{\text{exit}} \leftarrow \text{Sat}(S_{\text{patch},l}, \psi_{\text{patch},l})$
16: **for** $p \in \text{Reach}^*(l, S') \cap \text{Exit}(S')$ **do**
17: $Q \leftarrow \texttt{FindMatches}(p, S_{\text{exit}})$
18: **for** $((p, w), p', (p', w'))$ **in** Q **do**
19: $E(M_{\text{new}}) \leftarrow E(M_{\text{new}}) \cup (p', w)$
20: $\text{Cond}(p', w) \leftarrow \texttt{CondAll}$
21: $\text{Cond}(p', w') \leftarrow \texttt{CondAnyNot}$
22: **end for**
23: **end for**
24: **for** p' **in** $S_{\text{patch},l}$ **do**
25: **if** $\exists e' \in \text{In}(p'), i \in I'_{s,g} \,.\, h(e')g(p') \models \psi_{s,g_i}$ **then**
26: $\text{Rule}(p') \leftarrow \texttt{RuleSetIfSat}$
27: **else**
28: $\text{Rule}(p') \leftarrow \texttt{NULL}$
29: **end if**
30: **end for**
31: **end for**
32: **for** w **in** $\text{Reg}(S, \{\zeta_i\})$ **do**
33: **delete** w **from** S_{new} //…and dependent edges
34: **end for**
35: **return** M_{new}

from which the more general case can be composed: an execution path that enters then leaves a patch automaton in finitely many steps; or an execution path that enters but does not leave a patch automaton.

For the former case, let ρ be an execution path that enters and leaves in finitely many steps a patch automaton with index l. Recall the new strategy M_{new} is an automaton with finite memory in which an execution path cannot leave one of the sets of patch nodes $S_{\text{patch},l}$ unless all system goals satisfied in S' are satisfied. By the fact that $M_{\text{patch},l}$ realizes specification φ_l, $S_{\text{patch},l}$ can eventually be departed for any admissible environment behavior.

Given all locally satisfiable system goals have been visited at least once, the execution path ρ returns to the original automaton at a node v that was previously reachable from the same node at which the patch set $S_{\text{patch},l}$ was entered, l. This follows from the local navigation goals in $\psi_{\text{patch},l}$, which were selected to have the same valuation as reachable exit points in M, i.e. $\text{Reach}^*(l, S') \cap \text{Exit}(S')$.

The environment could have driven the system from l to v under the original strategy M. Hence, any valuations on the execution trace after the path visits patch set $S_{\text{patch},l}$ were reachable in the original M. Therefore subsequent infinite-often satisfaction of system goals is preserved in this case under the new strategy M_{new}.

The latter case is only possible for those $M_{\text{patch},l}$ where $\text{Reach}^*(l, S') \cap \text{Exit}(S') = \varnothing$, i.e. for which the original

Fig. 4. `FindTrans`(). Subroutine to find relevant safety properties.

$I'_{\alpha,t} \leftarrow \varnothing$
for i **in** $I_{\alpha,t}$ **do**
  if $\exists x \in \Sigma_{\mathcal{X}_{\mathrm{inc}}}, y \in \Sigma_{\mathcal{Y}_{\mathrm{inc}}}$. $xy \not\models \psi_{\alpha,t_i}$ **then**
    $I'_{\alpha,t} \leftarrow I'_{\alpha,t} \cup \{i\}$
  end if
end for
return $I'_{\alpha,t}$

Fig. 5. `LocalGoals`(). Subroutine to find goals locally satified in original strategy M.

$I'_{\alpha,g} \leftarrow \varnothing$
for i **in** $I_{\alpha,g}$ **do**
  if $\mathrm{Sat}(S, \psi_{\alpha,g_i}) \cap \mathrm{Reg}(S, B_{r_{\mathrm{inc}}}(\zeta_i)) \neq \varnothing$ **then**
    $I'_{\alpha,g} \leftarrow I'_{\alpha,g} \cup \{i\}$
  end if
end for
return $I'_{\alpha,g}$

Fig. 6. `FindMatches`(). Subroutine to find nodes with matching valuations and outward edges.

Require: l node, and S set of nodes
Match $\leftarrow \varnothing$
for w **in** S **do**
  if $\exists e_1 \in \mathrm{In}(l), e_2 \in \mathrm{In}(w)$. $h(e_1)g(l) = h(e_2)g(w)$ **then**
    for e'_1 **in** $\mathrm{Out}(l)$ **do**
      for e'_2 **in** $\mathrm{Out}(w)$ **do**
        if $h(e'_1) = h(e'_2)$ **then**
          Match $\leftarrow$ Match $\cup \{(e'_1, w, e'_2)\}$
        end if
      end for
    end for
  end if
end for
return Match

Fig. 7. `RuleClear`(). Clear all memory.

Require: Mem finite memory
for ξ **in** Mem **do**
  $\xi \leftarrow 0$
end for
return Mem

Fig. 8. `RuleSetIfSat`(). Set any memory entry for which corresponding system goal is true.

Require: memory Mem, node p, local goal index set $I'_{s,g}$
for i **in** $I'_{s,g}$ **do**
  if $\exists e \in \mathrm{In}(p)$. $h(e)g(p) \models \psi_{s,g_i}$ **then**
    $\xi_i \leftarrow 1$
  end if
end for
return Mem

Fig. 9. `CondAll`(). Enable transition only if all memory entries are True.

if $\forall \xi \in \mathrm{Mem}$. $\xi = 1$ **then**
  return True
else
  return False
end if

Fig. 10. `CondAnyNot`(). Enable transition only if at least one memory entry is False

if $\exists \xi \in \mathrm{Mem}$. $\xi = 0$ **then**
  return True
else
  return False
end if

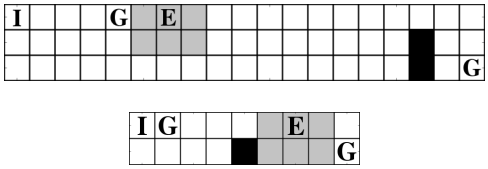

Fig. 11. Gridworld hallways. "**I**" is the robot initial position, each location marked with "**G**" must be visited infinitely often (like surveillance), and the shaded cell contains an obstacle that can move freely within it. The original maps, used for synthesizing nominal controllers, do *not* include the black walls.

solution M would have a continuous trajectory forever remaining in N. It follows that $I'_{s,g} = I_{s,g}$. Because $M_{\mathrm{patch},l}$ realizes φ_l by hypothesis, therefore the execution path forever remaining in $S_{\mathrm{patch},l}$ must be correct with respect to the global specification Eq. (2). ■

If the structure of the continuous space changes as considered here, e.g., newly sensed unreachability of cell $\zeta_i \in Z/\mathcal{P}$, then some safety properties of the original specification may need to be updated to reflect this. Therefore, soundness is with respect to the updated global specification.[1]

Remark 4: The worst-case (asymptotic) time and memory complexity of the presented algorithm is that of global re-synthesis.

V. Examples

Results below were obtained and displayed using the tools TuLiP [19] and Gephi [20]. Software implementing our algorithm and the gridworld setting will soon be released open source.

Consider the gridworld shown in upper Fig. 11. The robot begins operation in the upper-left cell and must visit the cells marked "**G**" infinitely often. Also it must avoid a dynamic obstacle in the shaded cells. Despite its apparent simplicity, the problem is actually quite large when *all* movements of the obstacle and robot are addressed. Recall that we seek to *guarantee* correctness of the synthesized controller, so probabilities of success are not sufficient.

The nominal strategy M has 259 nodes. After patching, the new control strategy M_{new} has 1036 nodes and is depicted in Fig. 12. The final solution consists of nonlocal parts of M that remain valid, patches $M_{\mathrm{new},l}$, and edges connecting them. Control flow is clockwise about the top

[1]Nonetheless with some abuse of terminology, we use "original" and "global" interchangeably.

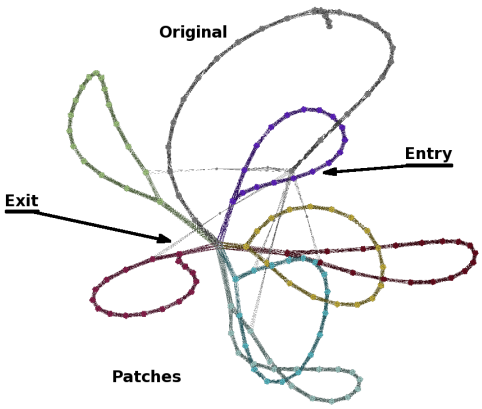

Fig. 12. Patched automaton M_{new} for problem in upper Fig. 11. The loops indicate the robot visiting both goals infinitely often. The knots are clusters of nodes, representing execution path variations given different obstacle (environment) positions. The small tail toward the top of the figure is for initialization. Execution traces run there for a finite prefix before proceeding into the main behaviors.

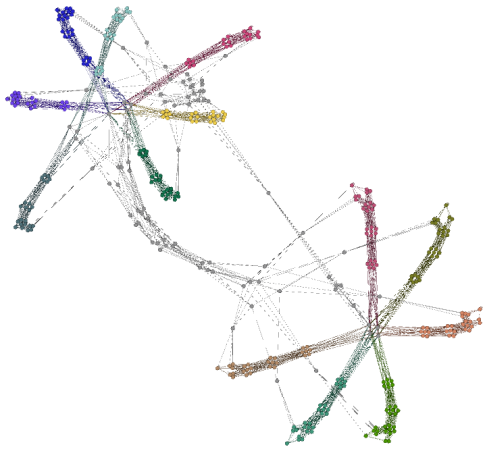

Fig. 13. Patched automaton M_{new} for problem in lower Fig. 11. The fan-like parts in the upper-left and lower-right are patches, and the web-like subgraphs are the remaining valid parts of the nominal strategy.

of the figure. An execution path proceeds over this "hoop" before branching into one of the patches. After the rightmost goal in the gridworld is satisfied as a local system goal in φ_l, the execution path leaves the patch and returns to what remains of M around the part noted by "Exit" in the figure.

A similar example is shown in lower Fig. 11, where now the robot must correctly negotiate a dynamic obstacle to satisfy both recurring goals. The result of patching is depicted in Fig. 13.

VI. Conclusion

We presented a method for using online sensor data to update a robot controller locally while recovering global correctness. Our approach focuses on reachability of the continuous state space. Such changes are *static*, e.g., when a mobile robot must update its floor layout given new mapping data. Future work will address changes in dynamic properties of the environment estimated online.

Acknowledgments

The authors gratefully acknowledge Eric Wolff and Sandeep Chinchali for comments on this manuscript. This work is partially supported by the Boeing Corporation.

References

[1] 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. [Online]. Available: http://doi.acm.org/10.1145/75277.75293

[2] A. Church, "Logic, arithmetic, and automata," in *Proceedings of the International Congress of Mathematicians*, August 1962, pp. 23–35.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proceedings of the 2009 IEEE Conference on Decision and Control (CDC)*, 2009, pp. 2222–2229.

[9] 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.

[10] B. Johnson and H. Kress-Gazit, "Probabilistic analysis of correctness of high-level robot behavior with sensor error," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[11] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta, "Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees," in *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*, May 2010, pp. 3227–3232.

[12] X. C. Ding, S. L. Smith, C. Belta, and D. Rus, "LTL control in uncertain environments with probabilistic satisfaction guarantees," in *Proceedings of 18th IFAC World Congress*, 2011.

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

[14] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, pp. 872–923, May 1994.

[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] 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/

[17] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

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

[19] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "TuLiP: A software toolbox for receding horizon temporal logic planning," in *Proceedings of 14th International Conference on Hybrid Systems: Computation and Control (HSCC'11)*, 2011. [Online]. Available: http://tulip-control.sourceforge.net

[20] http://gephi.org/, "Gephi: open-source software for network visualization and analysis," September 2011.