On the Revision Problem of Specification Automata

Kangjin Kim, Georgios E. Fainekos and Sriram Sankaranarayanan

***Abstract*— One of the important challenges in robotics is the automatic synthesis of provably correct controllers from high level specifications. One class of such algorithms operates in two steps: (i) high level discrete controller synthesis and (ii) low level continuous controller synthesis. In this class of algorithms, when phase (i) fails, then it is desirable to provide feedback to the designer in the form of revised specifications that can be achieved by the system. In this paper, we address the minimal revision problem for specification automata. That is, we construct automata specifications that are as "close" as possible to the initial user intent, by removing the minimum number of constraints from the specification that cannot be satisfied. We prove that the problem is computationally hard and we encode it as a satisfiability problem. Then, the minimal revision problem can be solved by utilizing efficient SAT solvers.**

I. INTRODUCTION

One of the fundamental challenges in robotics is how to achieve fully automatic correct-by-design synthesis of control software. Scalable methods and techniques for synthesizing correct by construction controllers can potentially revolutionize the design process, yielding huge benefits in terms of improved safety, reliability and time to market. Moreover, governmental agencies will have the tools to certify in short time medical robotic devices, autonomous automobiles and airplanes etc.

Currently, the automatic synthesis problem for complex dynamical systems is broken into two levels. First, high level synthesis of the discrete controller and, subsequently, low level composition of simple control laws. A variety of such theories and methodologies exist which are usually classified based on the high level specification framework. For example, a very general and well developed framework for high-level programming is the extended Motion Description Language (MDLe) [1] which has interesting composition properties [2]. A more recent attempt to use Context Free Languages (CFL) for robotic control appears in [3]. Finite automata are used as a specification language in [4]. Another very popular formal specification language is temporal logics with multiple applications in robotics [5]–[12].

Nevertheless, one issue that has not been adequately addressed so far in such combined high-low level controller synthesis frameworks is what happens when the high-level synthesis phase fails. That is, when the specification cannot be realized in the current environment under the current system dynamics, then the current high-level synthesis frameworks simply report a failure. Thus, the user is usually left in the dark as of why the specification failed and, most importantly, on what the system can actually achieve that is close to the initial intentions of the user.

This work has been partially supported by award NSF CNS 1116136.

K. Kim and G. Fainekos are with the School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281, USA {Kangjin.Kim,fainekos}@asu.edu

S. Sankaranarayanan is with the Department of Computer Science, University of Colorado, Boulder, CO srirams@colorado.edu

In this paper, we study the theoretical foundations of the specification revision problem when both the system and the specification can be represented by ω-automata [13]. In particular, we focus on the Minimal Revision Problem (MRP), i.e., finding the *closest* satisfiable specification to the initial specification, and we prove that the problem is NP-complete. In view of this negative result, we study whether encoding MRP as a satisfiability problem and utilizing state-of-the-art satisfiability solvers provides an efficient solution to the problem.

The specification revision problem for automata based planning techniques is a relatively new problem. In our previous work [14], we introduced the specification revision problem for Linear Temporal Logic (LTL). There, we identified conditions such that the minimal revision can be efficiently solved and we provided a randomized algorithm to return some specification revision (but not necessarily the minimal). Finding out why a specification is not satisfiable on a model is a problem that is very related to the problems of *vacuity* and *coverage* in model checking [15]. Another related problem is the detection of the causes of unrealizability in LTL games. In this case, a number of heuristics have been developed in order to localize the error and provide meaningful information to the user for debugging [16], [17]. Along these lines, LTLMop [18] was developed to debug unrealizable LTL specifications in reactive planning for robotic applications.

II. PROBLEM FORMULATION

In this paper, we work with discrete abstractions (Finite State Machines) of the continuous robotic control system [5]. This is a common practice in approaches that hierarchically decompose the control synthesis problem into high level discrete planning synthesis and low level continuous feedback controller composition [5], [6], [11]. Each state of the Finite State Machine (FSM) $\mathcal{T}$ is labeled by a number of symbols from a set $\Pi = \{\pi_0, \pi_1, \ldots, \pi_n\}$ that represent regions in the workspace of the robot or, more generally, in its configuration space (see [19] for precise definitions of workspace and configuration spaces). The control requirements for such a system can be posed using specification automata $\mathcal{B}$ with Büchi acceptance conditions [13] also known as ω-automata.

The following example, which is the running example of this paper, presents such a typical scenario for motion

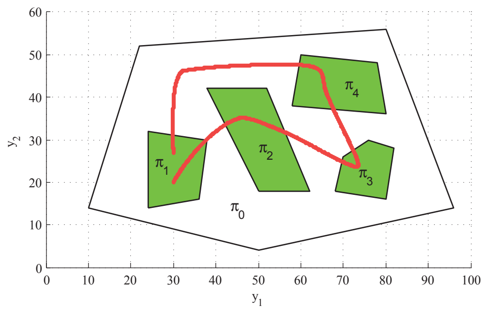

Fig. 1. The simple environment of Example 1 along with a low speed mobile robot trajectory that satisfies the specification.

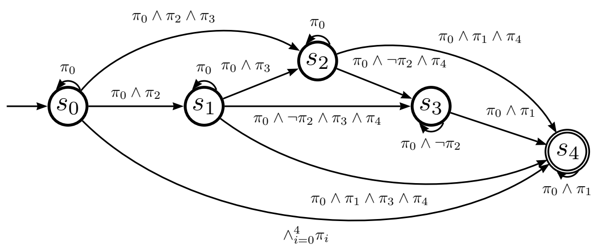

Fig. 2. The specification automaton of Example 1.

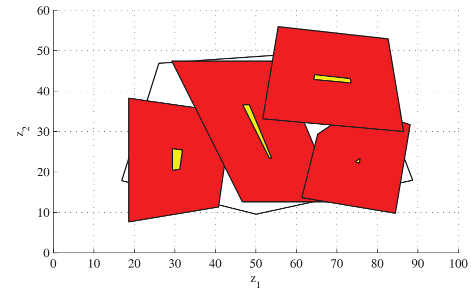

Fig. 3. The modified environment of Fig. 1 under large bounds on the permissible acceleration U. The dark gray regions indicate areas that should be avoided in order to satisfy $\neg\pi_i$ while the light gray regions indicate areas that should be visited in order to satisfy π_i.

planning of a mobile robot.

Example 1 (Robot Motion Planning): We consider a mobile robot which operates in a planar environment. The continuous state variable $x(t)$ models the internal dynamics of the robot whereas only its position $y(t)$ is observed. In this paper, we will consider a 2nd order model of the motion a planar robot (dynamic model):

$$\begin{aligned} \dot{x}_1(t) &= x_2(t), & x_1(t) \in \mathbb{R}^2,\ x_1(0) \in X_{1,0} \\ \dot{x}_2(t) &= u(t), & x_2(t) \in \mathbb{R}^2,\ x_2(0) = 0,\ u(t) \in U \\ y(t) &= x_1(t). \end{aligned}$$

The robot is moving in a convex polygonal environment π_0 with four areas of interest denoted by $\pi_1, \pi_2, \pi_3, \pi_4$ (see Fig. 1). The robot is placed somewhere in the region labeled by π_1. The robot must accomplish the task: "Stay always in π_0 and visit area π_2, then area π_3, then area π_4 and, finally, return to and stay in region π_1 while avoiding area π_2," which is captured by the specification automaton in Fig. 2.

In [5], we developed a hierarchical framework for motion planning for dynamic models of robots. The hierarchy consists of a high level logic planner that solves the motion planning problem for a kinematic model of the robot, e.g.,

$$\dot{z}(t) = u(t),\ y'(t) = z(t),\ z(t) \in \mathbb{R}^2,\ z(0) \in Z_0.$$

Then, the resulting hybrid controller is utilized for the design of an approximate tracking controller for the dynamic model. Since the tracking is approximate, the sets that the atomic propositions map to need to be modified (see Fig. 3) depending on the maximum speed of the robot so the that the controller has a guaranteed tracking performance. For example, in Fig. 3, the regions that now must be visited are the contracted light gray regions, while the regions to be avoided are the expanded dark gray regions. However, the set modification might make the specification unrealizable. E.g., in Fig. 3, the robot cannot move from π_4 to π_1 while avoiding π_2, even though the specification can be realized on the workspace of the robot that the user perceives. In this case, the user is entirely left in the dark as of why the specification failed and, more importantly, on what actually the system can achieve under these new constraints. $\triangle$

When a specification $\mathcal{B}$ is not satisfiable on a particular system $\mathcal{T}$, then the current motion planning and control synthesis methods [5], [6] based on automata theoretic concepts simply return that the specification is not satisfiable without any other user feedback. The goal of this paper is to study specification feedback mechanisms when the automata theoretic planning phase fails to return a plan.

Problem 1 (Minimal Revision Problem (MRP)): Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}$, if the specification $\mathcal{B}$ cannot be satisfied on $\mathcal{T}$, then find the "closest" specification $\mathcal{B}'$ to $\mathcal{B}$ which can be satisfied on $\mathcal{T}$.

Problem 1 was first introduced in [14] for Linear Temporal Logic (LTL) specifications. In [14], we provided solutions to the debugging and (not minimal) revision problems and we demonstrated that we can easily get a minimal revision of the specification when the discrete controller synthesis phase fails due to unreachable states in the system. Thus, in this paper, we concentrate on the harder problem of minimal revision when all the states on $\mathcal{T}$ are reachable.

Assumption 1: All the states on $\mathcal{T}$ are reachable.

In this paper, we prove that if the specification is provided as an arbitrary ω-automaton, then a restricted version of the minimal revision problem is NP-complete. This theoretical result has profound implications on two fronts.

- First, the LTL MRP is most likely NP-complete as well. The ω-automata that correspond to LTL formulas are a restricted subset of all possible ω-automata. However, the structural properties that cause the NP-completeness of the problem exist in this restricted class of automata as well.
- Second, we now know that in order to find polynomial time solutions to Problem 1, we will either have to develop randomized algorithms (as we did in [14]) or develop approximation algorithms.

III. Constructing Discrete Controllers

In this section, we provide a brief review of the automata based motion planning. This is required in order to understand the new contributions of this paper. In order to use ω-automata to specify requirements for continuous systems, we need to construct a finite partition of the robot's workspace [19]. For that purpose, we can use many efficient cell decomposition methods for polygonal environments [19]. This results in a topological graph $G = (Q, E)$ which describes which cells are topologically adjacent, i.e., each node $q \in Q$ in the graph represents a cell and each edge $e = (q, q') \in E$ in the graph implies topological adjacency of the cells. Each such cell will be a state in the FSM which will be labeled by one or more atomic propositions from Π. Next, we formally define the FSM that can be constructed from the graph G.

Definition 1 (FSM): A Finite State Machine is a tuple $\mathcal{T} = (Q, Q_0, \to_\mathcal{T}, h_\mathcal{T}, \Pi)$ where: Q is a set of states; $Q_0 \subseteq Q$ is the set of possible initial states; $\to_\mathcal{T} = E \subseteq Q \times Q$ is the transition relation; and, $h_\mathcal{T} : Q \to \mathcal{P}(\Pi)$ maps each state q to the set of atomic propositions that are true on q.

We define a *path* on the FSM to be a sequence of states and a *trace* to be the corresponding sequence of sets of propositions. Formally, a path is a function $p : \mathbb{N} \to Q$ such that for each $i \in \mathbb{N}$ we have $p(i) \to_\mathcal{T} p(i+1)$ and the corresponding trace is the function composition $\bar{p} = h_\mathcal{T} \circ p : \mathbb{N} \to \mathcal{P}(\Pi)$. The language $\mathcal{L}(\mathcal{T})$ of $\mathcal{T}$ consists of all possible traces.

In this work, we are interested in the ω-automata that will impose certain requirements on the traces of $\mathcal{T}$. ω-automata differ from the classic finite automata in that they accept infinite strings (traces of $\mathcal{T}$ in our case).

Definition 2: A automaton is a tuple $\mathcal{B} = (S_\mathcal{B}, s_0^\mathcal{B}, \Omega, \delta_\mathcal{B}, F_\mathcal{B})$ where: $S_\mathcal{B}$ is a finite set of states; $s_0^\mathcal{B}$ is the initial state; Ω is an input alphabet; $\delta_\mathcal{B} : S_\mathcal{B} \times \Omega \to \mathcal{P}(S_\mathcal{B})$ is a transition function; and $F_\mathcal{B} \subseteq S_\mathcal{B}$ is a set of final states.

When $s' \in \delta_\mathcal{B}(s, l)$, we also write $s \xrightarrow{l}_\mathcal{B} s'$ or $(s, l, s') \in \to_\mathcal{B}$. A *run* r of $\mathcal{B}$ is a sequence of states $r : \mathbb{N} \to S_\mathcal{B}$ that occurs under an input trace $\bar{p}$ taking values in Ω. That is, for $i = 0$ we have $r(0) = s_0^\mathcal{B}$ and for all $i \geq 0$ we have $r(i) \xrightarrow{\bar{p}(i)}_\mathcal{B} r(i+1)$. Let $\lim(\cdot)$ be the function that returns the set of states that are encountered infinitely often in the run r of $\mathcal{B}$. Then, a run r of an automaton $\mathcal{B}$ over an infinite trace $\bar{p}$ is *accepting* if and only if $\lim(r) \cap F_\mathcal{B} \neq \emptyset$. This is called a Büchi acceptance condition. Finally, we define the language $\mathcal{L}(\mathcal{B})$ of $\mathcal{B}$ to be the set of all traces $\bar{p}$ that have a run that is accepted by $\mathcal{B}$.

A *specification* automaton is an automaton with Büchi acceptance condition where the input alphabet is the powerset of the labels of the system $\mathcal{T}$, i.e., $\Omega = \mathcal{P}(\Pi)$. In order to simplify the discussion in Section IV, we will be using the following assumptions and notation

- we define the set $E_\mathcal{B} \subseteq S_\mathcal{B}^2$, such that $(s, s') \in E_\mathcal{B}$ iff $\exists l \in \Omega$, $s \xrightarrow{l}_\mathcal{B} s'$; and,
- we define the function $\lambda_\mathcal{B} : S_\mathcal{B}^2 \to \Omega$ which maps a pair of states to the label of the corresponding transition, i.e., if $s \xrightarrow{l}_\mathcal{B} s'$, then $\lambda_\mathcal{B}(s, s') = l$; and if $(s, s') \notin E_\mathcal{B}$, then $\lambda_\mathcal{B}(s, s') = \emptyset$.

In brief, our goal is to generate paths on $\mathcal{T}$ that satisfy the specification $\mathcal{B}_\mathbf{s}$. In automata theoretic terms, we want to find the subset of the language $\mathcal{L}(\mathcal{T})$ which also belongs to the language $\mathcal{L}(\mathcal{B}_\mathbf{s})$. This subset is simply the intersection of the two languages $\mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{B}_\mathbf{s})$ and it can be constructed by taking the product $\mathcal{T} \times \mathcal{B}_\mathbf{s}$ of the FSM $\mathcal{T}$ and the specification automaton $\mathcal{B}_\mathbf{s}$. Informally, the automaton $\mathcal{B}_\mathbf{s}$ restricts the behavior of the system $\mathcal{T}$ by permitting only certain acceptable transitions. Then, given an initial state in the FSM $\mathcal{T}$, we can choose a particular trace from $\mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{B}_\mathbf{s})$ according to a preferred criterion.

Definition 3: The product automaton $\mathcal{A} = \mathcal{T} \times \mathcal{B}_\mathbf{s}$ is the automaton $\mathcal{A} = (S_\mathcal{A}, s_0^\mathcal{A}, \mathcal{P}(\Pi), \delta_\mathcal{A}, F_\mathcal{A})$ where:

- $\mathcal{S}_\mathcal{A} = Q \times S_{\mathcal{B}_\mathbf{s}}$,
- $s_0^\mathcal{A} = \{(q_0, s_0^{\mathcal{B}_\mathbf{s}}) \mid q_0 \in Q_0\}$,
- $\delta_\mathcal{A} : S_\mathcal{A} \times \mathcal{P}(\Pi) \to \mathcal{P}(S_\mathcal{A})$ s.t. $(q_j, s_j) \in \delta_\mathcal{A}((q_i, s_i), l)$ iff $q_i \to_\mathcal{T} q_j$ and $s_j \in \delta_{\mathcal{B}_\mathbf{s}}(s_i, l)$ with $l \subseteq h_\mathcal{T}(q_j)$,
- $F_\mathcal{A} = Q \times F$ is the set of accepting states.

Note that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{B}_\mathbf{s})$. We say that $\mathcal{B}_\mathbf{s}$ is *satisfiable* on $\mathcal{T}$ if $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Moreover, finding a satisfying path on $\mathcal{T} \times \mathcal{B}_\mathbf{s}$ is an easy algorithmic problem [20]. First, we convert automaton $\mathcal{T} \times \mathcal{B}_\mathbf{s}$ to a directed graph and, then, we find the strongly connected components (SCC) in that graph. If at least one SCC that contains a final state is reachable from an initial state, then there exist accepting (infinite) runs on $\mathcal{T} \times \mathcal{B}_\mathbf{s}$ that have a finite representation. Each such run consists of two parts: a part that is executed only once (from an initial state to a final state) and a part that is repeated infinitely (from a final state back to itself). Note that if no final state is reachable from the initial or if no final state is within an SCC, then the language $\mathcal{L}(\mathcal{A})$ is empty and, hence, the high level synthesis problem does not have a solution. *Namely, the synthesis phase has failed and we cannot find a system behavior that satisfies the specification* $\mathcal{B}_\mathbf{s}$.

IV. The Specification Revision Problem

Intuitively, a revised specification is one that can be satisfied on the discrete abstraction of the workspace or the configuration space of the robot. In order to search for a minimal revision, we need first to define an ordering relation on automata as well as a distance function between automata. Similar to the case of LTL formulas in [14], we do not want to consider the "space" of all possible automata, but rather the "space" of specification automata which are semantically close to the initial specification automaton $\mathcal{B}_\mathbf{s}$. The later will imply that we remain close to the initial intention of the designer. We propose that this space consists of all the automata that can be derived from $\mathcal{B}_\mathbf{s}$ by removing atomic propositions from the transition input. Our definition of the ordering relation between automata relies upon the previous assumption.

Definition 4 (Relaxation): Let $\mathcal{B}_1 = (S_{\mathcal{B}_1}, s_0^{\mathcal{B}_1}, \mathcal{P}(\Pi), \to_{\mathcal{B}_1}, F_{\mathcal{B}_1})$ and $\mathcal{B}_2 = (S_{\mathcal{B}_2}, s_0^{\mathcal{B}_2}, \mathcal{P}(\Pi), \to_{\mathcal{B}_2}, F_{\mathcal{B}_2})$ be two specification automata. Then, we say that $\mathcal{B}_2$ is a relaxation of $\mathcal{B}_1$ and we write $\mathcal{B}_1 \preceq \mathcal{B}_2$ if and only if $S_{\mathcal{B}_1} = S_{\mathcal{B}_2} = S$, $s_0^{\mathcal{B}_1} = s_0^{\mathcal{B}_2}$, $F_{\mathcal{B}_1} = F_{\mathcal{B}_2}$ and

1) $\forall (s, l, s') \in \to_{\mathcal{B}_1} - \to_{\mathcal{B}_2} . \exists l'$. $(s, l', s') \in \to_{\mathcal{B}_2} - \to_{\mathcal{B}_1}$ and $l' \subseteq l$.
2) $\forall (s, l, s') \in \to_{\mathcal{B}_2} - \to_{\mathcal{B}_1} . \exists l'$. $(s, l', s') \in \to_{\mathcal{B}_1} - \to_{\mathcal{B}_2}$ and $l' \supseteq l$.

We remark that $\preceq$ is a partial order over specification automata. Also, if $\mathcal{B}_1 \preceq \mathcal{B}_2$, then $\mathcal{L}(\mathcal{B}_1) \subseteq \mathcal{L}(\mathcal{B}_2)$ since the relaxed automaton allows more behaviors to occur. It is possible that two automata $\mathcal{B}_1$ and $\mathcal{B}_2$ cannot be compared under relation $\preceq$. We can now define the set of automata over which we will search for a minimal solution that has nonempty intersection with the system.

Definition 5: Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_\mathbf{s}$, the set of *valid relaxations* of $\mathcal{B}_\mathbf{s}$ is defined as $\mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T}) = \{\mathcal{B} \mid \mathcal{B}_\mathbf{s} \preceq \mathcal{B} \text{ and } \mathcal{L}(\mathcal{T} \times \mathcal{B}) \neq \emptyset\}$.

We can now search for a minimal solution in the set $\mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$. That is, we can search for some $\mathcal{B} \in \mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$ such that if for any other $\mathcal{B}' \in \mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$, we have $\mathcal{B}' \preceq \mathcal{B}$, then $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}')$. However, this does not imply that a minimal solution semantically is minimal structurally as well. In other words, it could be the case that $\mathcal{B}_1$ and $\mathcal{B}_2$ are minimal relaxations of some $\mathcal{B}_\mathbf{s}$, and moreover, $\mathcal{B}_1$ requires the modification of only one transition while $\mathcal{B}_2$ requires the modification of two transitions. Therefore, we must define a distance on the set $\mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$, which accounts for the number of changes from the initial specification automaton $\mathcal{B}_\mathbf{s}$.

Definition 6: Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_\mathbf{s}$, we define the distance of any $\mathcal{B} \in \mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$ from $\mathcal{B}_\mathbf{s}$ to be $\mathbf{dist}_{\mathcal{B}_\mathbf{s}}(\mathcal{B}) = \sum_{(s,s') \in E_{\mathcal{B}_\mathbf{s}}} |\lambda_{\mathcal{B}_\mathbf{s}}(s, s') - \lambda_{\mathcal{B}}(s, s')|$ where $|\cdot|$ is the cardinality of the set.

Therefore, Problem 1 can be restated as:

Problem 2: Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_\mathbf{s}$ such that $\mathcal{L}(\mathcal{T} \times \mathcal{B}_\mathbf{s}) = \emptyset$, find $\mathcal{B} \in \arg\min\{\mathbf{dist}_{\mathcal{B}_\mathbf{s}}(\mathcal{B}') \mid \mathcal{B}' \in \mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})\}$.

A. Minimal Revision as a Graph Problem

We will solve Problem 2 by introducing Boolean variables that represent various possible revisions of the specification automaton $\mathcal{B}_\mathbf{s}$. Consequently, we extend the existing product automaton $\mathcal{T} \times \mathcal{B}_\mathbf{s}$ by adding edges labeled by a conjunction of Boolean revision variables that can enable the edges. The overall problem then becomes one of finding the least number of Boolean revision variables that need to be set to true so that the product graph has an accepting run.

Revision Variables: We first add Boolean revision variables $y(e_i, \pi_j)$ for each edge $e_i \in E_{\mathcal{B}_\mathbf{s}}$ and each atomic proposition $\pi_j \in \lambda_{\mathcal{B}_\mathbf{s}}(e_i)$ that labels the e_i transition on $\mathcal{B}_\mathbf{s}$. The revision variable proposes to relax the edge e_i by removing π_j from its set of atomic propositions. Let RevVars represent the set of all revision variables.

Graphs labeled with Revision Variables: We provide the formal definition of $G_\mathcal{A}$ which corresponds to a product automaton $\mathcal{A}$ while considering the effect of revisions.

Definition 7: Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_\mathbf{s}$, we define the graph $G_\mathcal{A} = (V, E, v_s, V_f, L)$, which corresponds to the product $\mathcal{A} = \mathcal{T} \times \mathcal{B}_\mathbf{s}$ as follows

- $V = \mathcal{S}$ is the set of nodes
- $E = E_\mathcal{A} \cup E_D \subseteq \mathcal{S} \times \mathcal{S}$, where $E_\mathcal{A}$ is the set of edges that correspond to transitions on $\mathcal{A}$, i.e., $((q, s), (q', s')) \in E_\mathcal{A}$ iff $\exists l \in \mathcal{P}(\Pi)$. $(q, s) \xrightarrow{l}_\mathcal{A} (q', s')$; and E_D is the set of edges that correspond to disabled transitions, i.e., $((q, s), (q', s')) \in E_D$ iff $q \to_\mathcal{T} q'$ and $s \xrightarrow{l}_{\mathcal{B}_\mathbf{s}} s'$ with $l \cap (\Pi - h_\mathcal{T}(q')) \neq \emptyset$.
- $v_s = s_0^\mathcal{A}$ is the source node,
- $V_f = F_\mathcal{A}$ is the set of sinks,
- $L : E \to \mathcal{P}(\text{RevVars})$ maps each edge of the graph with a set of revision variables that need to be set to true in order to enable it. The construction of the labeling function will be described subsequently.

We describe the construction of the labeling function $L : E \to \mathcal{P}(\text{RevVars})$ for the product graph $\mathcal{A}$. Let $e = ((q, s), (q', s'))$ be an edge in $\mathcal{A}$ corresponding to edge $e_i = (s, s')$ in $\mathcal{B}_\mathbf{s}$ and edge (q, q') in $\mathcal{T}$. Consider the set of atomic propositions given by $\Lambda(e) = \lambda_{\mathcal{B}_\mathbf{s}}(s, s') - h_\mathcal{T}(q')$. If $\Lambda(e) \neq \emptyset$, then it specifies those atomic propositions in $\lambda_{\mathcal{B}_\mathbf{s}}(s, s')$ that need to be removed in order to enable the edge in the product state. The label for the edge $e = ((q, s), (q', s'))$ is defined as: $L(e) = \{y((s, s'), \pi_j) \mid \pi_j \in \Lambda(e)\}$

B. Paths on Graphs labeled with Boolean Variables

We now present the problem of finding accepting paths on Boolean labeled graphs. Let $Y = \{y_1, \ldots, y_m\}$ be a set of Boolean variables and $G : (V, E)$ be a graph with a labeling function $L : E \to \mathcal{P}(Y)$, wherein each edge $e \in E$ is labeled with a set of Boolean variables $L(e) \subseteq Y$. The label on an edge indicates that the edge is *enabled* iff all the Boolean variables on the edge are set to true. Let $v_0 \in V$ be a marked initial state and $F \subseteq V$ be a set of marked final vertices.

Problem 3 (Minimal Accepting Path (MAP)): Inputs: A set of Boolean variables Y, graph G with edge labeling function L, initial vertex v_0 and final vertices $F \subseteq V$.

Output: A set $Z \subseteq Y$ of minimal cardinality such that setting all variables in Z to true and $Y - Z$ to false enables a path from v_0 to some final vertex $v_f \in F$ along with a cycle from v_f back to itself.

Theorem 1: Given an instance of the minimal accepting path problem (Y, G, L, v_0, F) and a bound W, the decision of problem of whether there exists a truth assignment $Z \subseteq Y$ such that $|Z| \leq W$ is NP-Complete.

C. MAP Encoding Into SAT

We discuss a SAT-based encoding of the minimal accepting path. Our encoding converts the search for a minimal truth assignment to a *pseudo-Boolean* optimization problem.

Let (Y, G, L, v_0, F) be a given instance of the minimal accepting path problem, wherein the graph G has vertices V and edges $E \subseteq V \times V$. Our goal is to first produce a Boolean formula $\Psi[Y, R]$ over the Boolean variables in Y and auxiliary variables in R (described below), such that for any truth assignment to the variables in Y, there is an accepting path iff $(\exists R)\Psi[Y, R]$.

The variables in R are of the form $\text{REACH}(v_0, v)$ and $\text{REACH}(v_f, v)$ for every vertex $v \in V$ and $v_f \in F$. The proposition $\text{REACH}(v_0, v)$ denotes that vertex v is reachable from v_0. Similarly for $v_f \in F$, the proposition $\text{REACH}(v_f, v)$ denotes that vertex v is reachable from v_f. We will discuss the encoding of reachability in terms of a Boolean formula. The encoding consists of many parts that are conjoined together (using the AND operator) to create the final formula involving variables in Y along with variables in R.

a) Reachability from v_0 and from $v_f \in F$: A node is reachable iff one of its predecessors is reachable and the Boolean condition on the edge holds. We assert the following clauses (assuming that $v_0 \neq v_f$, otherwise only either the clauses (v_0, v) or the clauses (v_f, v) need to be considered):

1) If $(v_0, v) \in E$, then $\text{REACH}(v_0, v) \Leftrightarrow \bigwedge_{y \in L(v_0,v)} y$.
2) If $(v_0, v) \notin E$, then

$$\text{REACH}(v_0, v) \Leftrightarrow \bigvee_{(u,v)\in E} \left(\text{REACH}(v_0, u) \wedge \bigwedge_{y\in L(u,v)} y \right).$$

3) If $(v_f, v) \in E$, then $\text{REACH}(v_f, v) \Leftrightarrow \bigwedge_{y \in L(v_f,v)} y$.
4) If $(v_f, v) \notin E$, then

$$\text{REACH}(v_f, v) \Leftrightarrow \bigvee_{(u,v)\in E} \left(\text{REACH}(v_f, u) \wedge \bigwedge_{y\in L(u,v)} y \right).$$

b) "Lasso" condition: Finally, we assert the existence of a final state which is reachable from itself:

$$\bigvee_{v_f \in F} (\text{REACH}(v_0, v_f) \wedge \text{REACH}(v_f, v_f))$$

SMT solvers such as Yices (http://yices.csl.sri.com) and Z3 (http://research.microsoft.com/projects/z3) allow us to search for a minimum weight satisfiable by specifying weights for setting a variable y to true.

V. Numerical Experiments

In this section, we demonstrate the application of our framework on our motivating example and, then, we assess the feasibility of posing MRP as a satisfiability problem. For the experiments, we utilized the ASU supercomputing center which consists of clusters of Dual 4-core processors, 16GB Intel(R) Xeon(R) CPU X5355 @2.66 GHz. Our implementations do not utilize the parallel architecture. The clusters were used to run the many different test cases in parallel on a single core. The operating system is CentOS release 5.5.

Example 2: First, we revisit our motivating Example 1. For this example, we used MiniSat [21] for solving the SAT encoding of MAP. The environment of Fig. 1 was abstracted into a state machine with 17 states. Thus, the graph $G_{\mathcal{A}}$ had 85 states and 140 atomic propositions: 10 on the specification automaton (5 positive + 5 negative) times 14 transitions on the specification automaton. The real running time was $11.7\,\text{sec}$ and our implementation returned the 13 revisions. The minimal 3 revisions were: (1) $y((s_3, s_3), \neg\pi_2)$, (2) $y((s_2, s_3), \pi_4)$, (3) $y((s_2, s_4), \pi_4)$.

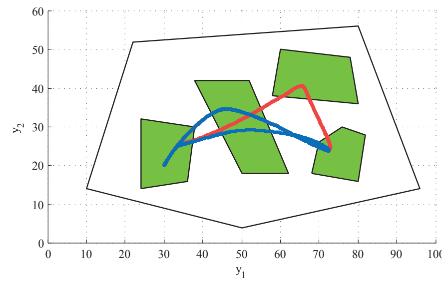

Fig. 4. The simple environment of Example 1 along with two trajectories generated using the revised specifications.

Revision (1) is the one that conforms the most with the human intuition of what a revision to the requirement "Stay always in π_0 and visit area π_2, then area π_3, then area π_4 and, finally, return to and stay in region π_1 while avoiding area π_2," should be. Namely, if there cannot be a solution that avoids π_2, then go through π_2. The motion generated under Revision (1) appears in Fig. 4 in light gray.

Revisions (2) and (3) actually generate the same behavior (see dark gray trajectory in Fig. 4). Namely, the resulting trajectory does not visit π_4 and does not avoid π_2. To see why this occurs, let us first consider revision (3). After visiting π_3 the specification automaton is in state s_2. Thus, now, we have the option to take transition (s_2, s_4) to get to region π_1 without having to avoid region π_2. On the other hand, Revision (2) permits the specification automaton to stay in state s_3 until the robot passes over region π_2 at which point the transition (s_2, s_3) can be taken.

We see that there are several minimal revisions some of which generate different behaviors. Thus, a feedback system to the user must supply many such different revisions and let the user select one of them.

We remark that our alternative implementation of the SAT encoding of MAP using Answer Set Programming (ASP) and ClaspD 1.1 [22] returned Revision (2) (along with 3 more non-minimal revisions) in less than 1 sec. △

To evaluate if the solution to the SAT encoding of the minimal revision problem can be solved efficiently we run a number of experiments. The SAT encoding implementation was performed using Answer Set Programming (ASP) [22] under ClaspD 1.1. We repeated each experiment many times and we report the minimum, maximum and average real running time. Since in some cases the computation time exceeded the 2hr hard bound that we had set, we also report the number of tests that succeeded out of the total number of trials. Note that the average value is reported for the test cases that succeeded in computing a minimal revision.

Table I compares the total number of nodes vs the total number of edges in a graph. For each pair of values we generated a random graph where about 20% of nodes are final and the number of atomic propositions is fixed. Each experiment was executed for a number of nodes and for a sparse graph, a medium connected graph and a dense graph. Each graph is generated randomly by providing the number of nodes, the number of edges, the number of atomic

Edges →	Sparse: $2n-2$				Medium: $3n$				Dense: n^2			
Nodes n ↓	min	avg	max	succ	min	avg	max	succ	min	avg	max	succ
10	0.0	0.1	0.2	100/100	0.0	0.0	0.1	100/100	0.0	0.1	0.9	100/100
100	0.3	0.6	1.5	100/100	0.9	41.5	1934.2	100/100	1425.1	2541.5	5970.4	67/100
200	1.8	4.7	24.1	100/100	9.5	273.4	6400.8	77/100				0/100
300	5.9	15.4	76.3	100/100	34.8	536.5	5624.3	71/100				0/100
400	14.7	58.2	244.9	100/100	87.1	1218.8	4175.3	50/100				0/100
500	33.2	125.7	473.0	100/100	176.8	1800.8	6939.2	48/100				0/100

TABLE I

NUMERICAL EXPERIMENTS: NUMBER OF NODES VERSUS NUMBER OF EDGES. THE REPORTED NUMBERS ARE MINIMUM, AVERAGE AND MAXIMUM RUNNING TIME IN SECONDS AND THE NUMBER OF TRIALS THAT SUCCESSFULLY COMPLETED WITHIN 2HR. FOR EACH RANDOMLY GENERATED GRAPH, THERE WERE n ATOMIC PROPOSITIONS.

propositions and the number of final states.

The experimental results indicate that a specification feedback and revision framework based on satisfiability solvers will be efficient only for small sized problems. The class of mission and motion planning problems that would generate graph sizes that can be solved efficiently within our framework is task planning for a single mobile robot within small - but complicated - environments such as an office building.

VI. CONCLUSIONS

In this paper, we introduced the problem of minimal revision of specification automata. Namely, if the specification for a task of a robot is provided as an ω-automaton and the specification cannot be satisfied on the model of the system, then propose a new specification automaton which defines requirements that can be satisfied on the system. The challenge in proposing a new specification automaton is that the new automaton should be as close as possible to the initial intent of the user. We proved that actually the minimal revision problem for specification automata is NP-complete. We also provided an encoding of the problem as a satisfiability problem which can be solved by the state-of-art satisfiability solvers. Even though our current solution is efficient for single robot scenarios, we expect that polynomial-time approximation or randomized algorithms will provide efficient solutions for multi-robot scenarios. This is the topic of our on-going research.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their detailed comments and suggestions.

REFERENCES

[1] D. Hristu-Varsakelis, M. Egerstedt, and P. S. Krishnaprasad, "On the complexity of the motion description language MDLe," in *Proceedings of the 42nd IEEE CDC*, December 2003, pp. 3360–3365.

[2] W. Zhang and H. G. Tanner, "Composition of motion description languages," in *Hybrid Systems: Computation and Control*, ser. LNCS, vol. 4981. Springer, 2008, pp. 570–583.

[3] N. Dantam and M. Stilman, "The motion grammar: Linguistic perception, planning, and control." in *Robotics: Science and Systems*, 2011.

[4] M. Karimadini and H. Lin, "Decomposability of global tasks for multi-agent systems," in *in Proc. of the 49th IEEE CDC*, 2010.

[5] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, Feb. 2009.

[6] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2010.

[7] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on LTL specifications," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Dec. 2004.

[8] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *International Conference on Robotics and Automation*. IEEE, 2010, pp. 2689–2696.

[9] S. Karaman, R. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," in *IEEE CDC*, 2008.

[10] P. Roy, P. Tabuada, and R. Majumdar, "Pessoa 2.0: a controller synthesis tool for cyber-physical systems," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*, ser. HSCC '11. New York, NY, USA: ACM, 2011, pp. 315–316.

[11] 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.

[12] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. New York, NY, USA: ACM, 2010, pp. 101–110.

[13] J. R. Buchi, "Weak second order arithmetic and finite automata," *Zeitschrift für Math. Logik und Grundlagen Math.*, vol. 6, 1960.

[14] G. E. Fainekos, "Revising temporal logic specifications for motion planning," in *Proceedings of the IEEE Conference on Robotics and Automation*, May 2011.

[15] O. Kupferman, W. Li, and S. A. Seshia, "A theory of mutations with applications to vacuity, coverage, and fault tolerance," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*. IEEE Press, 2008, pp. 25:1–25:9.

[16] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev, "Diagnostic information for realizability," in *Verification, Model Checking, and Abstract Interpretation*, ser. LNCS, F. Logozzo, D. Peled, and L. Zuck, Eds. Springer, 2008, vol. 4905, pp. 52–67.

[17] R. Konighofer, G. Hofferek, and R. Bloem, "Debugging formal specifications using simple counterstrategies," in *Formal Methods in Computer-Aided Design*. IEEE, Nov. 2009, pp. 152 –159.

[18] V. Raman and H. Kress-Gazit, "Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP," in *23rd International Conference on Computer Aided Verification*, ser. LNCS, vol. 6806. Springer, 2011, pp. 663–668.

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

[20] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: MIT Press, 1999.

[21] N. Sorensson and N. Een, "Minisat v1.13: A sat solver with conflict-clause minimization," in *In Proc. of SAT Competition: Solver Description*, 2005.

[22] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub, "Conflict-driven disjunctive answer set solving," in *Proceedings of the 11th International Conference on PKRR*, G. Brewka and J. Lang, Eds. AAAI Press, 2008, pp. 422–432.