Revising Temporal Logic Specifications for Motion Planning

Georgios E. Fainekos

Abstract— **In this paper, we introduce the problem of automatic formula revision for Linear Temporal Logic (LTL) motion planning specifications. Namely, if a specification cannot be satisfied on a particular environment, our framework returns information to the user regarding (i) why the specification cannot be satisfied and (ii) how the specification can be modified so it can become satisfiable. This work contributes towards rendering temporal logic motion planning frameworks more user friendly by providing feedback to the user when the LTL planning phase fails.**

I. Introduction

During the last several years, there has been an explosion of research that tries to bridge high level planning frameworks based on temporal logics with low level continuous control primitives for autonomous robotics applications (see [1]–[8] and the references therein). Temporal logics and, in general, formal languages provide a mathematical framework that facilitates reasoning about the correctness of synthesis of hybrid systems such that complex system requirements can be met. Essentially such complex specifications extend well beyond traditional control requirements into a holistic system design. The popularity of temporal logics over other formalisms, e.g., regular languages, can be attributed mainly to their resemblance to natural language. That property alone makes temporal logics good candidates for expressing complex system requirements. Along these lines, one can even develop computational interfaces between natural language and temporal logics [9], [10]. Therefore, temporal logics appear to be a suitable medium for our daily discourse with future autonomous robots.

Nevertheless, one issue that has not been addressed so far in the automata theoretic temporal logic planning frameworks is what happens when the high-level planning phase fails. That is, when the temporal logic specification cannot be realized in the current environment under the current system dynamics, then the high-level synthesis framework simply reports a failure. In detail, in temporal logic motion planning frameworks such as in [1]–[3], the motion planning problem is divided into two subproblems. First, the high level planning problem is solved and, then, local feedback controllers are composed. In detail, the high-level planning phase can be reduced to a double nested Depth or Breadth-First Search on a graph. If a goal state in the graph is not reachable from the start state, then a plan that satisfies the specification does not exist. Unfortunately, the user is left in the dark as of why the specification failed. Was the failure due to unreachable parts of the environment? Was it due to some logical inconsistencies in the specification? Was it due to the system dynamics?

In order to develop a truly user friendly temporal logic planning system, we need to provide feedback to the user when the planning stage fails. Or even better, we need to recommend to the user specifications that are satisfiable on our particular system and environment. Of course, such recommendations cannot be arbitrary formulas which happen to be satisfiable on our particular model (i.e., environment and system). For example, it is easy to see that the specification "always true" is a valid formula which is true on every possible model. Hence, these new specification recommendations must be as close as possible to the original intentions of the user.

In this paper, we provide a solution to exactly this problem. That is, our foundational contribution is the definition of the Linear Temporal Logic (LTL) formula revision problem. For the solution of this problem, we define a partial order over LTL specifications and we try to choose a formula that relaxes the initial specification such that it is satisfiable on the model and, at the same time, it is a minimal element in the set of all satisfiable formulas on the model.

To the authors' best knowledge, this problem has not been discussed before in the literature. The closest related research problem is query checking [11], [12]. In query checking, given a model of the system and a temporal logic formula ϕ, some subformulas in ϕ are replaced with placeholders. Then, the problem is to determine a set of Boolean formulas such that if these formulas are placed into the placeholders, then ϕ holds on the model. The problem of LTL revision as defined here is substantially different from query checking. For one, the user does not know where to position the placeholders in the formula when the planning fails.

Another related problem is the problem of revising a system model such that it satisfies a temporal logic specification [13], [14]. Note that in this paper, we are trying solve the opposite problem, i.e., we are trying to relax the temporal logic specification such that it can be realized on the system. The main motivation for our work is that the model of the system, i.e., the environment and the system dynamics, cannot be modified and, therefore, we need to understand what we can be achieved with the current constraints.

II. Problem Formulation

In this work, we model the motion of the mobile robot within its workspace using Finite State Machines (FSM) [15]. This is a common practice in approaches that hierarchically decompose the motion planning problem into high

This work has been supported by an ASU startup fund.

G. Fainekos is with the School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281, USA fainekos@asu.edu

level discrete planning synthesis and low level continuous feedback controller composition [1], [3], [7]. Each state of the FSM $\mathcal{T}$ is labeled by a symbol from a set $\Pi = \{\pi_0, \pi_1, \ldots, \pi_n\}$ that represents a region in the robot's workspace.

The user requirements or specifications are expressed as Linear Temporal Logic (LTL) formulas [15]. In order to make apparent the use of LTL for the composition of temporal specifications, we first give an informal description of the traditional and temporal operators. LTL formulas are built over a set of atoms, the set Π in our case, using combinations of the traditional and temporal operators. Traditional logic operators are the *conjunction* ($\wedge$), *disjunction* ($\vee$) and *negation* ($\neg$). Some of the temporal operators are *eventually* ($\Diamond$), *always* ($\Box$), *until* ($\mathcal{U}$) and *release* ($\mathcal{R}$). LTL can describe the usual properties of interest for control problems, i.e. *reachability* ($\Diamond\pi$) and *safety:* ($\Box\pi$ or $\Box\neg\pi$). Beyond the usual properties, LTL can capture sequences of events and certain infinite behaviors. For example, for repeatedly visiting regions π_1, π_2 and π_3 in that order, we can write $\Box\Diamond(\pi_1 \wedge \Diamond(\pi_2 \wedge \Diamond\pi_3))$. Examples of more complicated specifications can be found in [1], [3].

When a specification ϕ is not satisfiable on a particular system $\mathcal{T}$, then the current LTL motion planning [1], [3] and control synthesis methods [16], [17] 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 develop methods for user feedback when the automata theoretic LTL planning phase fails to return a plan.

Problem 1 (LTL Debugging and Revision): Given a system $\mathcal{T}$ and an LTL formula ϕ, if the specification ϕ cannot be satisfied on $\mathcal{T}$, then return to the user:

1) **[Debugging]** information on how the planning fails;
2) **[Revision]** a formula ψ which can be satisfied on $\mathcal{T}$;
3) **[Minimal Revision]** if possible, the closest formula ψ to ϕ which can be satisfied on $\mathcal{T}$.

Debugging has a straightforward solution and we are simply going to make some remarks on our proposed solution in the following discussion. However, the revision problem is substantially more involved. In this paper, we make the following fundamental contributions:

- First, we provide a formal definition of what it means for two specifications to be close.
- Then, based on this definition, we provide algorithms that automatically modify the initial specification ϕ and provide as feedback to the user a new specification ψ which can be realized on the system.
- We prove that ψ is a minimal formula when the planning phase fails due to unreachable states in $\mathcal{T}$.
- Finally, we pose an open problem regarding the efficient computation of minimal revision in the general case.

III. Temporal Logic Motion Planning

In this section, we provide a brief review of the automata based Linear Temporal Logic (LTL) [15] motion planning. This is required in order to understand the new contributions of this paper. Further details on automata based LTL planning can be found in [1], [17], [18].

In order to use discrete logics to reason about continuous systems, we need to construct a finite partition of the robot's workspace[1]. 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 FTS which will be labeled by one or more atomic propositions from Π. Next, we formally define the FTS that can be constructed from the graph G.

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

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

We now formally introduce LTL without the next time operator as a specification language for defining the desired robot behavior. We assume that we use only formulas in Negation Normal Form (NNF) and that each negated atomic proposition $\neg\pi$ has been replaced by a new symbol, e.g. $\overline{\pi}$, which is added to Π. The details on why LTL in NNF is equivalent to full LTL can be found in [1]. The use of LTL in NNF greatly simplifies the technical results. However, in general, formulae examples are easier to understand in full LTL. Thus, in the following examples we will freely use the negation operator.

Definition 2 (LTL Syntax): The set $LTL(\Pi)$ of all LTL formulas built over a set of atomic propositions Π is defined recursively as $\phi ::= \pi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \Diamond\phi_1 \mid \phi_1 \mathcal{U} \phi_2 \mid \Box\phi_1 \mid \phi_1 \mathcal{R} \phi_2$ for $\pi \in \Pi$ and $\phi_1, \phi_2 \in LTL(\Pi)$.

In the following, we let $(\bar{p}, i) \models \phi$ denote the satisfiability of an LTL formula ϕ over a trace $\bar{p}$ starting at time i. We define the language $\mathcal{L}(\phi)$ to be the set of all traces that satisfy ϕ at time 0, i.e., $\mathcal{L}(\phi) = \{\bar{p} \in \mathcal{P}(\Pi)^\omega \mid (\bar{p}, 0) \models \phi\}$.

Definition 3 (LTL Semantics): The semantics of any LTL formula $\phi \in LTL(\Pi)$ is defined as (for $i, j \in \mathbb{N}$):

$(\bar{p}, i) \models \top, \quad (\bar{p}, i) \not\models \bot, \quad (\bar{p}, i) \models \pi$ iff $\pi \in \bar{p}(i)$

$(\bar{p}, i) \models \phi_1 \wedge \phi_2$ if $(\bar{p}, i) \models \phi_1$ and $(\bar{p}, i) \models \phi_2$

$(\bar{p}, i) \models \phi_1 \vee \phi_2$ if $(\bar{p}, i) \models \phi_1$ or $(\bar{p}, i) \models \phi_2$

$(\bar{p}, i) \models \phi_1 \mathcal{U} \phi_2$ if there exists $j \geq i$ such that $(\bar{p}, j) \models \phi_2$ and for all k with $i \leq k < j$ we have $(\bar{p}, k) \models \phi_1$

$(\bar{p}, i) \models \phi_1 \mathcal{R} \phi_2$ if for all $j \geq i$ we have $(\bar{p}, j) \models \phi_2$ or there exists $k \in [i, j)$ such that $(\bar{p}, k) \models \phi_1$

[1]Similarly, such a finite partition can be constructed on the state space of the system rather than the workspace. However, this only affects the resulting FTS and, hence, the results in this paper still apply.

In this work, we are interested in the construction of automata that only accept the traces of $\mathcal{T}$ which satisfy the LTL formula ϕ. Such automata (which are referred to as Büchi automata [15]) differ from the classic finite automata in that they accept infinite strings (traces of $\mathcal{T}$ in our case).

Definition 4 (Automaton): A Büchi automaton is a tuple $\mathcal{B} = (S_\mathcal{B}, s_{0\mathcal{B}}, \Omega, \lambda_\mathcal{B}, F_\mathcal{B})$ where: $S_\mathcal{B}$ is a finite set of states; $s_{0\mathcal{B}}$ is the initial state; Ω is an input alphabet; $\lambda_\mathcal{B} : S_\mathcal{B} \times \Omega \to \mathcal{P}(S_\mathcal{B})$ is a transition relation; and $F_\mathcal{B} \subseteq S_\mathcal{B}$ is a set of final states.

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 $r(i+1) \in \lambda_\mathcal{B}(r(i), \bar{p}(i))$. 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 a Büchi automaton $\mathcal{B}$ over an infinite trace $\bar{p}$ is *accepting* if and only if $\lim(r) \cap F_\mathcal{B} \neq \emptyset$. 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 $\mathcal{B}$.

For each LTL formula ϕ, we can construct a Büchi automaton $\mathcal{B}_\phi = (S_{\mathcal{B}_\phi}, s_{0\mathcal{B}_\phi}, \mathcal{P}(\Pi), \lambda_{\mathcal{B}_\phi}, F_{\mathcal{B}_\phi})$ that accepts the infinite traces which satisfy the specification ϕ, i.e., $\mathcal{L}(\mathcal{B}_\phi) = \mathcal{L}(\phi)$. The translation from an LTL formula ϕ to a Büchi automaton $\mathcal{B}_\phi$ is a well studied problem and, thus, we refer the reader to [15] and the references therein for the theoretical details behind this translation.

Now that all the related terminology is defined, we can give an overview of the basic steps involved in the temporal logic planning [18]. In brief, our goal is to generate paths on $\mathcal{T}$ that satisfy the specification ϕ. In automata theoretic terms, we want to find the subset of the language $\mathcal{L}(\mathcal{T})$ which also belongs to the language $\mathcal{L}(B_\phi)$. This subset is simply the intersection of the two languages $\mathcal{L}(\mathcal{T}) \cap \mathcal{L}(B_\phi)$ and it can be constructed by taking the product $\mathcal{T} \times B_\phi$ of the FTS $\mathcal{T}$ and the Büchi automaton B_ϕ. Informally, the Büchi automaton B_ϕ restricts the behavior of the system $\mathcal{T}$ by permitting only certain acceptable transitions. Then, given an initial state in the FTS $\mathcal{T}$, we can choose a particular trace from $\mathcal{L}(\mathcal{T}) \cap \mathcal{L}(B_\phi)$ according to a preferred criterion.

Definition 5: The product automaton $\mathcal{A} = \mathcal{T} \times B_\phi$ is the automaton $\mathcal{A} = (S_\mathcal{A}, s_{0\mathcal{A}}, \mathcal{P}(\Pi), \lambda_\mathcal{A}, F_\mathcal{A})$ where:

- $S_\mathcal{A} = Q \times S_{\mathcal{B}_\phi}$,
- $s_{0\mathcal{A}} = \{(q_0, s_{0\mathcal{B}_\phi}) \mid q_0 \in Q_0\}$,
- $\lambda_\mathcal{A} : S_\mathcal{A} \times \mathcal{P}(\Pi) \to \mathcal{P}(S_\mathcal{A})$ s.t. $(q_j, s_j) \in \lambda_\mathcal{A}((q_i, s_i), l)$ iff $q_i \to_\mathcal{T} q_j$ and $s_j \in \lambda_{B_\phi}(s_i, l)$ with $l \subseteq h_\mathcal{T}(q_j)$,
- $F_\mathcal{A} = Q \times F$ is the set of accepting states.

By construction, the following lemma is satisfied.

Lemma 1 (Adapted from [18]): A trace $\bar{p}$ of $\mathcal{T}$ that satisfies the specification ϕ exists iff the language of $\mathcal{A}$ is non-empty, i.e., $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{T}) \cap \mathcal{L}(B_\phi) \neq \emptyset$.

We say that ϕ is *satisfiable* on $\mathcal{T}$ if $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Moreover, finding a satisfying path on $\mathcal{T} \times B_\phi$ is an easy algorithmic problem [15]. First, we convert automaton $\mathcal{T} \times B_\phi$ 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 $\mathcal{T} \times B_\phi$ 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 $\mathcal{L}(\mathcal{T} \times B_\phi)$ is empty and, hence, the temporal logic planning problem does not have a solution. *Namely, the planning phase has failed and we cannot find a system behavior that satisfies the specification.*

IV. Revising LTL Specifications

In many cases, it is possible that the specification ϕ cannot be satisfied on the system $\mathcal{T}$. Then, the question that is raised is, if the specification ϕ is not satisfiable on our system, what is the "closest" related specification ϕ' that can be satisfied?

Being close to our initial specification ϕ is very important. If we relax our specification too much, for example, if we set $\phi' = \top$, then the specification becomes satisfiable on our system no matter what. On the other hand, if we choose an unrelated specification to ϕ which is satisfiable on our system, then we do not really achieve our initial design requirement. Therefore, we need first to define an ordering relation between specifications and, then, try to choose one that is as close as possible to the initial specification.

A. Closeness of LTL formulas

As a natural ordering relation between LTL formulas, we choose the set inclusion on the set of traces that they characterize. This is a natural choice since if a formula is not satisfiable over a system $\mathcal{T}$, then we should look for a specification that imposes less constraints on the system.

Definition 6: Let $\phi, \psi \in LTL(\Pi)$, then we define $\phi \preceq \psi$ if and only if $\mathcal{L}(\phi) \subseteq \mathcal{L}(\psi)$.

It is easy to see that $\preceq$ is a partial order on $LTL(\Pi)$. However, $(LTL(\Pi), \preceq)$ is not a lattice [20] since any two LTL formulas have several upper or lower bounds without a greatest or a least element. For example, the upper bound of $\{\pi_0, \Diamond\pi_1\}$ contains $\pi_0 \vee \Diamond\pi_1$ and $\bot \wedge (\pi_0 \vee \Diamond\pi_1)$. However, if we consider the quotient $[LTL(\Pi)]_{/=}$ of $LTL(\Pi)$ under the language equivalence relation $=$, then $([LTL(\Pi)]_{/=}, \preceq)$ becomes a lattice. E.g., since $\mathcal{L}(\pi_0 \vee \Diamond\pi_1) = \mathcal{L}(\bot \wedge (\pi_0 \vee \Diamond\pi_1))$, $\pi_0 \vee \Diamond\pi_1$ and $\bot \wedge (\pi_0 \vee \Diamond\pi_1)$ belong to the same equivalence class.

Remark 1: In the following, in order to reduce clutter in the notation, we will use $LTL(\Pi)$ to also refer to the quotient $[LTL(\Pi)]_{/=}$ and we will use ϕ to also refer to the equivalence class $[\phi]_{/=}$. The exact meaning will be clear from the context.

Therefore, abusing the notation, the top element in the lattice is the formula $\phi = \top$, while the bottom element is the formula $\phi = \bot$. If two formulas cannot be compared, i.e., we have $\phi_1 \not\preceq \phi_2$ and $\phi_1 \not\succeq \phi_2$, then we write $\phi_1 \parallel \phi_2$.

Now that we have defined an ordering between formulas, we can formally define what we are searching for. Essentially, we are looking for a new specification ϕ' that can be satisfied on $\mathcal{T}$ such that it is higher in the order than the initial specification ϕ and, at the same time, remains as close

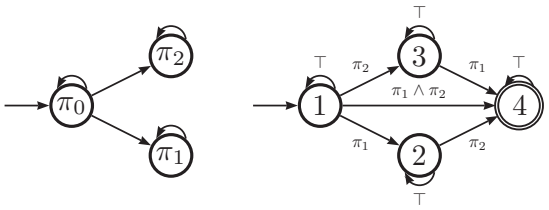

Fig. 1. The automaton of the environment $\mathcal{T}_1$ of example 1 and the Büchi automaton for the specification $\phi_1 = \diamond\pi_1 \wedge \diamond\pi_2$.

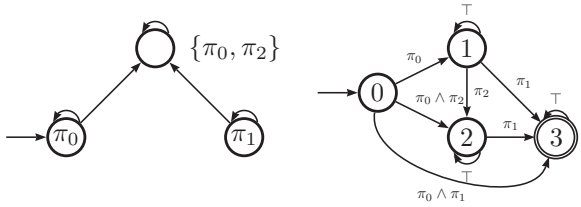

Fig. 2. The automaton of the environment $\mathcal{T}_2$ of Example 1 and the Büchi automaton for the specification $\phi_2 = \pi_0 \wedge \diamond(\pi_1 \vee (\pi_2 \wedge \diamond\pi_1))$.

to ϕ as possible. In other words, we try to minimally relax ϕ such that it can be satisfied on $\mathcal{T}$.

Example 1: Let us consider the specification $\phi_1 = \diamond\pi_1 \wedge \diamond\pi_2$ and the environment $\mathcal{T}_1$ of Fig. 1. It is apparent that there does not exist a plan that would make the specification satisfied on this particular environment since if you reach either the state π_1 or the state π_2, then you cannot go back and visit the other state also. In this case, the specification $\phi_1' = (\diamond\pi_1 \wedge \diamond\pi_2) \vee \pi_0$ is a minimally relaxed specification of ϕ_1, i.e., $\phi_1 \preceq \phi_1'$, which can be satisfied on $\mathcal{T}_1$. Similarly, $\diamond\pi_1$ and $\diamond\pi_2$ are minimally modified formulas of ϕ which are satisfiable on $\mathcal{T}$.

Several observations are in order. First, note that the formulas $\pi_0 \wedge \diamond\pi_1$ and $\diamond\Box\pi_1$ are also satisfiable on $\mathcal{T}_1$ and, furthermore, $\pi_0 \wedge \diamond\pi_1 \preceq \diamond\pi_1$ and $\diamond\Box\pi_1 \preceq \diamond\pi_1$. Nevertheless, $\phi_1 \not\preceq \pi_0 \wedge \diamond\pi_1$ and $\phi_1 \not\preceq \diamond\Box\pi_1$, which means that these specifications are not a relaxation of the initial requirement. Similarly, even though $\phi_1 \preceq \phi_1'$, ϕ_1' should not be included in our recommended possible formula revisions since it does not seem to be related to the initial intention of the user (due to the arbitrary disjunction with π_0). Second, notice that $\diamond\pi_1 \parallel \diamond\pi_2$. This implies that there can be several minimal solutions to consider from. Finally, it might be beneficial for the user to have the option to choose any possibility among the proposed solutions. In other words, the revised formula could be $\diamond\pi_1 \vee \diamond\pi_2$. □

Next, we formalize the intuition behind the preceding example. Let $\mathfrak{F}_\mathcal{T}$ be the set of all formulas that can be satisfied on $\mathcal{T}$, that is,

$$\mathfrak{F}_\mathcal{T} = \{\phi \in LTL(\Pi) \mid \mathcal{L}(\mathcal{T} \times \mathcal{B}_\phi) \neq \emptyset\}.$$

For a subset Φ of $LTL(\Pi)$, the upper bound of Φ is

$$\{\Phi\}^u = \{\psi \in LTL(\Pi) \mid \forall\phi \in \Phi.\phi \preceq \psi\}.$$

Namely, $\{\Phi\}^u$ contains all the LTL formulas which are "larger" than all the formulas in Φ. As Example 1 indicated, we would like to avoid entering unrelated atomic propositions in our revised specification. Therefore, given an LTL formula ϕ, we might want to restrict our search for a new LTL formula over the set $LTL(AP(\phi))$, where $AP(\phi)$ is the set of atomic propositions that appear in ϕ.

However, simply restricting the search for a new specification over the set of LTL formulas that are built over the atomic propositions that appear in the initial specification might not be enough (as Example 2 will indicate). Therefore, we also introduce the notion of *valid relaxations*. Informally, a valid formula relaxation is one that recursively relaxes each atomic proposition π of the initial specification ϕ.

Definition 7 (Valid Relaxation): Let $\phi \in LTL(\Pi)$, the set $\mathfrak{R}(\phi)$ of all valid formula relaxations of ϕ can be constructed using the recursive operator $\mathbf{rel}(\phi)$ as follows:

$$\mathbf{rel}(\pi) \in \{\pi\}^u \cap LTL(\{\pi\}) \text{ for } \pi \in \Pi$$
$$\mathbf{rel}(\phi_1 \text{ OP } \phi_2) = \mathbf{rel}(\phi_1) \text{ OP } \mathbf{rel}(\phi_2)$$

where OP is any Boolean or temporal operator.

Then, the following result is immediate.

Theorem 1: For any $\phi \in LTL(\Pi)$ and $\phi' \in \mathbf{rel}(\phi)$, we have $\phi \preceq \phi'$.

In our problem, given a formula ϕ such that $\mathcal{L}(\mathcal{T} \times \mathcal{B}_\phi) = \emptyset$, we are looking for a specification in the set $\mathfrak{F}_\mathcal{T}^\phi = \mathfrak{F}_\mathcal{T} \cap \{\phi\}^u \cap LTL(AP(\phi))$ or in the set $\mathfrak{RF}_\mathcal{T}^\phi = \mathfrak{F}_\mathcal{T}^\phi \cap \mathbf{rel}(\phi)$. Ideally, we would like to pick the smallest element in $\mathfrak{F}_\mathcal{T}^\phi$ (or $\mathfrak{RF}_\mathcal{T}^\phi$) under the order $\preceq$. As we pointed in Example 1, the set $\mathfrak{F}_\mathcal{T}^\phi$ (or $\mathfrak{RF}_\mathcal{T}^\phi$) does not necessarily have a least element. Therefore, we will focus our attention to minimal elements instead.

B. Formula Revision due to Unreachable States

An LTL plan generation might fail due to the existence of unreachable states in the discrete abstraction of the environment. Namely, the specification asks for a particular part of the environment to be reached; however, that part of the environment is disconnected from current position of the system. In this case, the formula can be easily revised using only syntactic modifications. However, minimality of the revised specification cannot always be achieved. It is better to describe the procedure through a simple example.

Example 2: Consider the specification $\phi_2 = \pi_0 \wedge \diamond(\pi_1 \vee (\pi_2 \wedge \diamond\pi_1))$ and the environment $\mathcal{T}_2$ in Fig. 2. By simply running a reachability algorithm on $\mathcal{T}_2$, we can determine the set of atomic propositions that may become true on $\mathcal{T}_2$, i.e., $\{\pi_0, \pi_2\}$. Therefore, we can syntactically replace the 2nd occurrence of π_1 in ϕ_2 with $\top$ and derive the specification $\phi_2' = \pi_0 \wedge \diamond(\pi_1 \vee \pi_2)$ that becomes satisfiable on $\mathcal{T}_2$. Note, however, that ϕ_2' is not minimal in $\mathfrak{F}_{\mathcal{T}_2}^{\phi_2}$. For example, $\phi_2'' = \pi_0 \wedge \diamond(\pi_1 \vee (\pi_2 \wedge (\diamond\pi_1 \vee \pi_0))$ is satisfiable on $\mathcal{T}_2$ and, also, $\phi_2'' \preceq \phi_2'$. □

Formally, assume that we have a set $\mathfrak{U}$ of unreachable atomic propositions. This can be easily computed by finding the set of reachable atomic propositions on the graph of the discrete abstraction of the environment. Consider the

following recursive function $\mathbf{rem}_{\mathfrak{U}} : LTL(\Pi) \rightarrow LTL(\Pi)$ that operates on an LTL formula ϕ and removes all the (positive) occurrences of atomic propositions in $\mathfrak{U}$ that appear in conjunctions (recall that no negation operator appears in our formulas):

$$\begin{aligned}
\mathbf{rem}_{\mathfrak{U}}(\pi) &= \pi \in \Pi \\
\mathbf{rem}_{\mathfrak{U}}(\text{NTO}(\pi_1) \wedge \text{NTO}(\pi_2)) &= \top \text{ if } \pi_1, \pi_2 \in \mathfrak{U} \\
\mathbf{rem}_{\mathfrak{U}}(\phi_1 \wedge \text{NTO}(\pi)) &= \mathbf{rem}_{\mathfrak{U}}(\phi_1) \text{ if } \pi \in \mathfrak{U} \\
\mathbf{rem}_{\mathfrak{U}}(\phi_1 \text{ OP } \phi_2) &= \mathbf{rem}_{\mathfrak{U}}(\phi_1) \text{ OP } \mathbf{rem}_{\mathfrak{U}}(\phi_2)
\end{aligned}$$

where OP is any Boolean or temporal operator, i.e., $\mathcal{U}$ or $\mathcal{R}$, and $\text{NTO}(\pi)$ is any nested temporal operator of zero depth and higher such that the right operant is only π, e.g., $\psi_1 \mathcal{U}(\psi_2 \mathcal{R} \pi)$ or, simply, π. Then, the following result is immediate.

Theorem 2: Let $\phi \in LTL(\Pi)$ be not satisfiable on FTS $\mathcal{T}$ and $\emptyset \neq \mathfrak{U} \subseteq \Pi$ be an unreachable set of labels in $\mathcal{T}$. Set $\phi' = \mathbf{rem}_{\mathfrak{U}}(\phi)$. Then, we have $\phi \preceq \phi'$. Moreover, ϕ' is minimal in $\mathfrak{RF}_{\mathcal{T}}^{\phi}$, i.e., if for any $\psi \in \mathfrak{RF}_{\mathcal{T}}^{\phi}$, we have $\psi \preceq \phi'$, then $\mathcal{L}(\psi) = \mathcal{L}(\phi')$.

C. Formula Revision due to Specification Inconsistencies

As opposed to unreachable states, logical inconsistencies cannot be efficiently resolved syntactically. Essentially, a syntactic modification algorithm would work by relaxing each subformula in the specification and, then, running the planning algorithm. It is evident that in order to find a minimal revision of the initial specification, we would need to consider all possible combinations of relaxing the subformulas. In the simplest case, we might need to run at least $O(2^{|AP(\phi)|})$ times the planning algorithm assuming that we are looking for a valid formula relaxation and that we maximally relax each atomic proposition, i.e., we set the atomic proposition to $\top$. Note that the above approach does not necessarily return a minimal formula even in the set $\mathfrak{RF}_{\mathcal{T}}^{\phi}$. When a state with label π is reachable, then there could be several candidates in $\{\pi\}^u$ for relaxing the requirement π.

Here, we provide an algorithm to relax the initial specification ϕ by using the synchronous product of the environment $\mathcal{T}$ with the specification $\mathcal{B}_\phi$. The intuition behind the algorithm is as follows. In Section III, we have already discussed that $\mathcal{L}(\mathcal{T} \times \mathcal{B}_\phi) = \emptyset$ if and only if there is no path from an initial state to a final state or from a final state back to itself. Our algorithm tries to permit more transitions on $\mathcal{B}_\phi$ such that a final state on $\mathcal{T} \times \mathcal{B}_\phi$ becomes reachable.

We first consider the finite part of the plan. Similarly to the planning case, we start a Depth First Search (DFS) from the initial state of $\mathcal{A} = \mathcal{T} \times \mathcal{B}_\phi$, but now we built the product automaton on-the-fly (similar to the nested DFS algorithm presented in [21]). Modifying the basic version of DFS for such an on-the-fly graph exploration only requires minor modifications and some additional book-keeping. As the algorithm explores the graph of $\mathcal{A}$, we mark the nodes which have transitions that are not allowed by the specification. When the algorithm has marked all the states as explored, but no final state on $\mathcal{A}$ has been reached, then we return to the latest visited state with forbidden transitions. We modify $\mathcal{B}_\phi$ so that it permits one of the forbidden transitions, and we continue the graph exploration. The process is repeated until we reach a final state $s_f \in F_{\mathcal{A}}$. Then, the same DFS algorithm is called starting from s_f and setting as a goal state s_f itself.

Algorithm 1 DFSModify

Input: System $\mathcal{T}$ and automaton $\mathcal{B}_\phi$ of specification ϕ.
Output: An automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{T} \times \mathcal{B}) \neq \emptyset$.

1: **procedure** DFSMODIFY($\mathcal{T}, \mathcal{B}_\phi$)
2: w_0.color $\leftarrow$ gray ▷ $w_0 = (q_0, s_0)$
3: List $\leftarrow \emptyset$, $w \leftarrow w_0$, $\mathcal{B} \leftarrow \mathcal{B}_\phi$
4: **while** Final state has not been reached **do**
5: **if** List is not empty **then**
6: $(s, s') \leftarrow$ last entry in List
7: Remove all pairs $(s, :)$ from the List
8: Relax one of the predicates on $s \rightarrow s'$ of $\mathcal{B}$
9: $w \leftarrow \arg\min\{w'.time \,|\, w' = (q, s), q \in Q\}$
10: $\forall w'$ s.t. w'.time $\geq w$.time, w'.color $\leftarrow$ white
11: **end if**
12: [Final State, List] $\leftarrow$ DFSVisit(w)
13: **end while**
14: **end procedure**

The LTL formula revision algorithm is presented in Algorithm 1. The DFSVisit in line 12 is the standard nested DFS algorithm [21] with the following modifications. First, the adjacency graph is randomly built on-the-fly since the user might be interested in generating different relaxations of the initial specification. Second, we check if we have reached a final state in $\mathcal{A}$ and, if so, we terminate and return. Finally, we maintain a List with the pairs of states (s, s') of $\mathcal{B}$ for which a transition on $\mathcal{T}$ is not allowed.

The new part in Algorithm 1 over the standard DFS is in the lines 6-10. Heuristically, we pick the last state of $\mathcal{B}$ in the List that had a forbidden transition on $\mathcal{T}$. This corresponds to a particular transition $s' \in \lambda_{\mathcal{B}}(s, l)$ for some set of atomic propositions $l \in \mathcal{P}(\Pi)$. We randomly pick some $\pi \in l$ and we either remove it if $|l| > 1$ or we replace it with $\top$ if $|l| = 1$. In the latter case, we allow any possible transition on $\mathcal{T}$. In line 9, we find the first state of $\mathcal{A}$ visited that had as component the state s of $\mathcal{B}$ under revision. This is required since revising a transition from s will affect the product construction of $\mathcal{T}$ with $\mathcal{B}$. Therefore, we need to re-initialize our search from the earliest such state. The need for using DFS over BFS becomes evident at this stage. DFS maintains the entry times to all the reachable vertices in the graph. Finally, in line 10, we make sure that all the states that were descendants of w are marked as not visited.

Theorem 3: Given a system $\mathcal{T}$ and specification automaton $\mathcal{B}_\phi$, Algorithm 1 always terminates and returns an automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{B}_\phi) \subseteq \mathcal{L}(\mathcal{B})$. The running time of DFSModify is $O(|S_{\mathcal{B}_\phi}| 2^{|\Pi|}(|S_{\mathcal{A}}| + |\lambda_{\mathcal{A}}|))$.

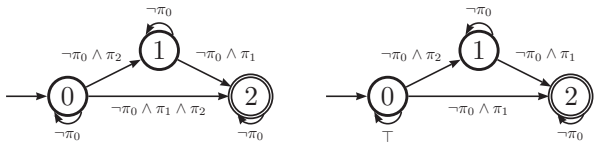

Fig. 3. Left: The Büchi automaton $\mathcal{B}_{\phi_3}$ from Example 3; Right: The automaton $\mathcal{B}_3$ that is returned by DFSModify.

Note that in practice the worst case running time is too pessimistic. First, it is highly unlikely that in practical specifications we will encounter outgoing transitions from each state labeled by all the subsets of Π. Second, it is also highly unlikely that each transition is labeled by the conjunction of all the atomic propositions in Π. Finally, we can safely assume that the desired initial specification ϕ is close to what can be achieved by the system. That is, a small number of changes should be sufficient to generate a specification that is satisfiable on the system. If this is not the case, then this implies that the initial specification is entirely irrelevant to what can be achieved by the system.

Example 3: Consider the system $\mathcal{T}_1$ in Fig. 1 and the specification $\phi_3 = \diamondsuit(\pi_2 \wedge \diamondsuit\pi_1) \wedge \square\neg\pi_0$. The Büchi automaton $\mathcal{B}_{\phi_3}$ that corresponds to ϕ_3 appears in Fig. 3. The automaton $\mathcal{B}_3$ that is returned by DFSModify also appears in Fig. 3. Note that the formula that corresponds to $\mathcal{B}_3$ is $\phi_3' = \diamondsuit(\pi_2 \wedge \diamondsuit\pi_1 \wedge \square\neg\pi_0) \vee \diamondsuit(\pi_1 \wedge \square\neg\pi_0)$. Finally, ϕ_3 is satisfiable on $\mathcal{T}_1$.

Problem 2 (Open): Up to this point, we have not addressed two issues. First, how do we translate the resulting automaton back to an LTL formula automatically? And second, are the formulas that correspond to the automata which are returned by DFSModify minimal?

It is known that alternating 1-weak Büchi automata (A1W) have the same expressive power with LTL [22]. Along these lines, Strejcek in his thesis [22] provides such an automatic translation from A1W to LTL. In the future, we plan to utilize such constructions in order to provide a completely automatic formula revision framework.

Regarding minimality, DFSModify performs a random search on the automaton graph. Therefore, the resulting specification automaton is not going to correspond to a minimal specification in the set $\mathfrak{F}_{\mathcal{T}}^{\phi}$, in general. Nevertheless, minimality might be achieved for certain fragments of LTL. In the future, we plan to explore such possibilities.

V. Conclusions and Future Work

We have presented the first steps toward a fully automatic Linear Temporal Logic (LTL) formula debugging and revision framework. This work contributes toward making temporal logics more user friendly for robotic applications. In the immediate future, we will be developing the theoretical framework of formula revision and debugging even further by addressing the open problems in Section IV-C as well as by exploring extensions to LTL games [2] and multi-robot scenarios [3].

Acknowledgements

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

References

[1] 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.

[2] 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.

[3] 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.

[4] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 61–71, 2007.

[5] 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.

[6] 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.

[7] 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.

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

[9] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Translating structured english to robot controllers," *Advanced Robotics Special Issue on Selected Papers from IROS 2007*, vol. 22, no. 12, pp. 1343–1359, 2008.

[10] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn, "What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution," in *Proceedings of the IEEE international conference on robotics and automation*, 2009.

[11] M. Chechik and A. Gurfinkel, "Tlqsolver: A temporal logic query checker," in *Proceedings of the 15th International Conference on Computer Aided Verification*, vol. 2725. Springer, 2003, pp. 210–214.

[12] G. Bruns and P. Godefroid, "Temporal logic query checking," in *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 2001, pp. 409 – 417.

[13] Y. Ding and Y. Zhang, "A logic approach for ltl system modification," in *15th International Symposium on Foundations of Intelligent Systems*, ser. LNCS, vol. 3488. Springer, 2005, pp. 435–444.

[14] M. Finger and R. Wassermann, "Revising specifications with CTL properties using bounded model checking," in *Brazilian Symposium on Artificial Intelligence*, ser. LNAI, vol. 5249, 2008, p. 157166.

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

[16] G. E. Fainekos, S. G. Loizou, and G. J. Pappas, "Translating temporal logic to controller specifications," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec. 2006, pp. 899–904.

[17] 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, 2008.

[18] G. D. Giacomo and M. Y. Vardi, "Automata-theoretic approach to planning for temporally extended goals," in *European Conference on Planning*, ser. LNCS, vol. 1809. Springer, 1999, pp. 226–238.

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

[20] G. E. Fainekos, "An introduction to multi-valued model checking," Dept. of CIS, Univ. of Pennsylvania, Tech. Rep. MS-CIS-05-16, September 2005.

[21] G. J. Holzmann, D. Peled, and M. Yannakakis, "On nested depth first search," in *Proceedings of the Second Workshop on the SPIN Verification System*, ser. DIMACS, vol. 32. American Mathematical Society, 1997.

[22] J. Strejcek, "Linear temporal logic: Expressiveness and model checking," Ph.D. dissertation, Masaryk University Brno, 2004.