Minimum-violation LTL Planning with Conflicting Specifications

Jana Tůmová Luis I. Reyes Castro Sertac Karaman Emilio Frazzoli Daniela Rus

***Abstract*— We consider the problem of automatic generation of control strategies for robotic vehicles given a set of high-level mission specifications, such as "Vehicle x must eventually visit a target region and then return to a base," "Regions A and B must be periodically surveyed," or "None of the vehicles can enter an unsafe region." We focus on instances when all of the given specifications cannot be reached simultaneously due to their incompatibility and/or environmental constraints. We aim to find the least-violating control strategy while considering different priorities of satisfying different parts of the mission. Formally, we consider the missions given in the form of linear temporal logic formulas, each of which is assigned a reward that is earned when the formula is satisfied. Leveraging ideas from the automata-based model checking, we propose an algorithm for finding an optimal control strategy that maximizes the sum of rewards earned if this control strategy is applied. We demonstrate the proposed algorithm on an illustrative case study.**

I. Introduction

Control strategy synthesis for robotic systems with high-level, complex, formally-specified goals has recently gained considerable attention in the robotics literature. A diverse set of techniques, including sampling and cell decomposition of the environment based on triangulations and rectangular partitions have been used to obtain discrete models of robotic systems; and a variety of temporal logics, including the Computation Time Logic (CTL) [20], Linear Temporal Logic (LTL) [4], [18], [19], [23], [24], and μ-calculus [15], [16] have been successfully utilized to express complex missions that arise in robotics applications. All these references focus on the *control synthesis* problem: find a control strategy that satisfies the given specification, if one exists; and report failure otherwise.

The usual execution of many robotic systems, however, involves cases when the mission specification cannot be satisfied as a whole. Yet, in most such examples, it is desirable to synthesize a control strategy that fulfills the most important rules, although by (temporarily) violating some of the less important ones. Consider, for example, an autonomous car navigating in urban traffic. The car must reach its final destination while abiding by the rules of the road, in particular, staying in the right lane and avoiding collision with obstacles. However, for this robot (and a human driver), it is more important not to collide with any other car or pedestrian, than to stay in its own lane. In fact the latter rule is temporarily violated, for instance, when taking over a parked car.

Another example is from the popular literature. Isaac Asimov's "three laws of robotics" (see [1]) defines how robots shall interact with humans. According to these laws, a robot may violate any order given by a human operator, if another human life comes in danger. Hence, the latter rule is issued a higher priority than the former one.

Motivated by these examples, in this paper, we consider the problem of *least-violating control synthesis*, *i.e.*, finding a control strategy that satisfies the most important pieces of the mission specification, even if the mission specification can not be fulfilled as a whole. The problem can be described as follows. Consider a deterministic transition system that models the robot and its environment. The states of the transition system may encode a select set of configurations of the robot (or n robots). Each state of the transition system is labeled with a set of atomic propositions. Examples for atomic propositions include "The robot is in a safe region," or "The first robot is in region A," etc. A list of mission specifications, including tasks that need to be fulfilled and rules that must be obeyed, is given in the form of linear temporal logic. Each specification in the list is assigned a priority. Roughly speaking, the least-violating synthesis problem is to find a trace over the transition system that satisfies as many high-priority tasks as possible.

Our work is related to [21], [22], where the authors study the following problem: given an LTL specification and a model of a robot that does not satisfy this specification, decide whether or not the invalidity is limited to the provided model. Related literature includes also [6], where the authors aim to pinpoint the (un)realizable fragments of the specification to reveal causes of the specification violation.]

Other related work includes the recent literature that aims to construct control strategies with minimal changes in the input. On one hand, in [9], [17], the authors aim to find a specification that (i) can be satisfied by the given model, and (ii) is close to the original specification according to a suitable metric. On the other hand, in [13] the author focuses on finding the least set of constraints (in the model) violating which results in the satisfaction of the specification and in [3], [5] the authors aim to repair model in the form of a transition system or a Markov chain in order to ensure the satisfaction a given CTL or PCTL formula, respectively.

Arguably, our work in this paper is closest to the one presented in [8], where the authors consider a transition system with the variables partitioned into control inputs for the car, controllable environment variables and disturbances.

This work is supported in part by Michigan/AFRL Collaborative Center on Control Sciences, AFOSR grant FA 8650-07-2-3744, the US National Science Foundation, grant CNS-1016213, the National Research Foundation of Singapore, through the Future Urban Mobility SMART IRG, and grants LH11065 and GAP202/11/0312 at Masaryk University.

J. Tůmová is with Masaryk University. L. I. Reyes Castro, S. Karaman, E. Frazzoli and D. Rus are with Massachusetts Institute of Technology (MIT). This work was initiated while the first author was visiting at MIT and Singapore-MIT Alliance for Research and Technology.

The mission specifications are captured as an ordered set of LTL formulas $\Phi = (\phi_1, \ldots, \phi_n)$. The goal is to find the maximal index $1 \leq m \leq n$ and a strategy for the robot ensuring the satisfaction of the subset of formulas $(\phi_1, \ldots, \phi_m)$ regardless of the environmental disturbances.

Variants of this problem have been addressed also from the perspective of control theory. For instance, in [7], the authors consider a system modeled as a Markov decision process and a set of specifications given in the form of Büchi automata, say $\mathcal{A}_1, \ldots, \mathcal{A}_n$, each of which is assigned a reward, say $rew_1, \ldots, rew_n$. They aim to find a strategy maximizing the total reward gained for the specifications weighted by the respective probabilities with which they are satisfied. The solution builds on translating the problem into a linear programming problem. Unfortunately, the time complexity of their algorithm is exponential in the size of the automata.

In contrast, our approach takes as input a deterministic transition system, a set of LTL formulas $\phi_1, \ldots, \phi_n$ with rewards $rew_1, \ldots, rew_n$, and we aim to construct a strategy maximizing the total reward gained for the specifications that are satisfied. We build the solution on the automata-based approach to model checking, which allows us to avoid exponential complexity (in the size of the input automata).

The contribution of this paper can be summarized as follows. We propose an algorithm for finding a least-violating trajectory, when the given specification can not be satisfied as a whole. As opposed to a "brute-force solution" enumerating all the possible subsets of specifications and attempting to find a strategy for each subset, we build our solution on a single control strategy synthesis procedure, thus substantially reducing the overall computational cost. We demonstrate the proposed approach in an illustrative example.

The rest of the paper is organized as follows. In Section II, we fix some necessary notation and preliminaries. In Section III, we introduce the problem and outline our approach to its solution. The solution, its correctness and complexity is then discussed in details in Section IV. Section V presents an illustrative case study and we conclude in Section VI.

II. Preliminaries

Given a set $\mathcal{S}$, let $|\mathcal{S}|$, $2^{\mathcal{S}}$, and $\mathcal{S}^\omega$ denote the cardinality of $\mathcal{S}$, the set of all subsets of S, and set of all infinite sequences of elements of $\mathcal{S}$, respectively. A finite and infinite sequence of elements of $\mathcal{S}$ is called a finite and infinite word over $\mathcal{S}$, respectively. Given a finite word w and a finite or an infinite word w' over $\mathcal{S}$, we use $w \cdot w'$ and $w^\omega = w \cdot w \cdot w \ldots$ to denote the word obtained by concatenation of w and w', and by infinitely many repetitions of w, respectively.

A. Model and Specification

Definition 1 (Transition System) *A* labeled deterministic transition system *is a tuple* $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$*, where* S *is a finite set of states;* $s_{init} \in S$ *is the initial state;* $\mathcal{R} \subseteq S \times S$ *is a deterministic transition relation;* Π *is a set of atomic propositions;* $\mathcal{L} : S \to 2^\Pi$ *is a labeling function.*

A *trace* of $\mathcal{T}$ is an infinite sequence of states $\tau = s_0 s_1 \ldots$ such that $s_0 = s_{init}$ and $(s_i, s_{i+1}) \in \mathcal{R}$, for all $i \geq 0$. A trace $\tau = s_0 s_1 \ldots$ produces a *word* $w(\tau) = \mathcal{L}(s_0)\mathcal{L}(s_1)\ldots$.

Definition 2 (Formulas of the LTL) *LTL formulas over the set* Π *of atomic propositions are constructed inductively according to the following rules:*

$$\phi ::= \top \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{X}\,\phi \mid \phi\,\mathsf{U}\,\phi,$$

where $\top$ *is a predicate that is always true,* $\pi \in \Pi$*,* $\neg$ *(negation) and* $\wedge$ *(conjunction) are standard Boolean operators and* X *(next) and* U *(until) are temporal operators.*

LTL formulas are interpreted over infinite words over 2^Π, such as those generated by the transition system from Def. 1. Informally speaking, the word $w = w(0)w(1)\ldots$ *satisfies* the atomic proposition π (denoted by $w \models \pi$), if π is satisfied in the first position of the word w, *i.e.*, if $\pi \in w(0)$. The formula $\mathsf{X}\,\phi$ states that ϕ holds in the following state. The formula $\phi_1\,\mathsf{U}\,\phi_2$ states that ϕ_2 is true eventually, and ϕ_1 is true at least until ϕ_2 is true. Furthermore, we define formulas $\mathsf{F}\,\phi \equiv \top\,\mathsf{U}\,\phi$ and $\mathsf{G}\,\phi \equiv \neg(\mathsf{F}\,\neg\phi)$ that state that ϕ holds *eventually* and *always*, respectively. LTL formulas can express various long term missions, including *surveillance* ($\mathsf{G}\,\mathsf{F}\,\phi$, always eventually visit ϕ), *global absence* ($\mathsf{G}\,\neg\psi$, globally avoid ϕ), *reactivity* ($\mathsf{G}\,\mathsf{F}\,\phi_1 \Rightarrow \mathsf{G}\,\mathsf{F}\,\phi_2$, if ϕ_1 holds infinitely often, then so must ϕ_2), among many others.

The language of all words that satisfy an LTL formula ϕ is denoted by $L(\phi)$. With a slight abuse of notation, we extend the satisfaction relation to traces of $\mathcal{T}$, *i.e.*, a trace τ *satisfies* ϕ (denoted by $\tau \models \phi$) if and only if the word w produced by τ satisfies ϕ. Similarly, a word w and a trace τ satisfies a set of formulas $\mathbf{\Phi}$ ($w \models \mathbf{\Phi}$ and $\tau \models \mathbf{\Phi}$) if and only if $w \models \phi$ and $\tau \models \phi$, for all $\phi \in \mathbf{\Phi}$, respectively.

Given a formula ϕ, we use $|\phi|$ to denote the *size* of the formula, *i.e.*, the number of operators present in ϕ, and we use $|\mathbf{\Phi}|$ to denote $\sum_{\phi \in \mathbf{\Phi}} |\phi|$.

Definition 3 (ω-Automaton) *An* ω*-automaton is a tuple* $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, Acc)$*, where* Q *is a finite set of states;* $q_{init} \in Q$ *is the initial state;* Σ *is an input alphabet;* $\delta \subseteq Q \times \Sigma \times Q$ *is a non-deterministic transition relation;* Acc *is the acceptance condition.*

The semantics of ω-automata are defined over infinite input words over Σ (such as those generated by transition system from Def. 1 if $\Sigma = 2^\Pi$). A *run* of the ω-automaton $\mathcal{A}$ *over* an input word $w = w(0)w(1)\ldots$ is a sequence of states $\rho = q_0 q_1 \ldots$, such that $q_0 = q_{init}$, and $(q_i, w(i), q_{i+1}) \in \delta$, for all $i \geq 0$. A *finite run* over a finite word $w_{\text{fin}} = w(0)\ldots w(l)$ is a finite sequence of states $\rho_{\text{fin}} = q_0 \ldots q_{l+1}$, such that $(q_i, w(i), q_{i+1})) \in \delta$, for all $i \in \{0, \ldots, l\}$.

A run $\rho = q_0 q_1 \ldots$ is *accepting* if it satisfies the acceptance condition Acc. For *Büchi automata (BA)*, Acc is a set of states $F \subseteq Q$, and ρ is accepting if it intersects F infinitely many times. For *generalized Büchi automata (GBA)*, the acceptance condition is a set of sets of states

$\mathcal{F} = \{F_1, \ldots, F_m\} \subseteq 2^Q$ and ρ is accepting if it intersects F_i infinitely many times for all $F_i \in \mathcal{F}$. A word w is *accepted* by $\mathcal{A}$ if there exists an accepting run over w. The *language* of all words accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

An ω-automaton is *non-blocking* if for all $q \in Q, \sigma \in \Sigma$ there exists $q' \in Q$, such that $(q, \sigma, q') \in \delta$. For each ω-automaton $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, Acc)$ a language equivalent non-blocking ω-automaton can be constructed simply by adding a new state q_{new} to Q and introducing a transition (q, σ, q_{new}) for all $q \in Q \cup \{q_{new}\}, \sigma \in \Sigma$, satisfying the property that $(q, \sigma, q') \notin \delta$ for all $q' \in Q$.

Definition 4 (GBA to BA) *A generalized Büchi automaton $\mathcal{G} = (Q_\mathcal{G}, q_{init,\mathcal{G}}, \Sigma, \delta_\mathcal{G}, \mathcal{F} = \{F_1, \ldots, F_m\})$, can be translated into a Büchi automaton $\mathcal{B} = (Q, q_{init}, \Sigma, \delta, F)$, such that $L(\mathcal{B}) = L(\mathcal{G})$ as follows: $Q = Q_\mathcal{G} \times \{1, \ldots, m\}$; $q_{init} = (q_{init,\mathcal{G}}, 1)$; $F = F_1 \times \{1\}$; and $\big((q, j), \sigma, (q', j')\big) \in \delta$ if and only if $(q, \sigma, q') \in \delta_\mathcal{G}$, and*

- *$q \notin F_j$ and $j' = j$, or*
- *$q \in F_j$ and $j' = (j \mod m) + 1$.*

Definition 5 (Automata Intersection) *Given n Büchi automata $\mathcal{B}_1, \ldots, \mathcal{B}_k$ where $\mathcal{B}_i = (Q_i, q_{init,i}, \Sigma, \delta_i, F_i)$ for all $1 \leq i \leq n$, a Büchi automaton $\mathcal{B} = (Q, q_{init}, \Sigma, \delta, F)$, such that $L(\mathcal{B}) = L(\mathcal{B}_1) \cap \ldots \cap L(\mathcal{B}_n)$ can be built as follows: $Q = Q_1 \times \ldots \times Q_n \times \{1, \ldots, n\}$; $q_{init} = (q_{init,1}, \ldots, q_{init,n}, 1)$; $F = F_1 \times Q_2 \times \ldots \times Q_n \times \{1\}$; and $\big((q_1, \ldots, q_n, j), \sigma, (q'_1, \ldots, q'_n, j')\big) \in \delta$ if and only if $(q_i, \sigma, q'_i) \in \delta_i$, for all $i \in \{1, \ldots, n\}$, and*

- *$q_j \notin F_j$ and $j' = j$, or*
- *$q_j \in F_j$ and $j' = (j \mod n) + 1$.*

Intuitively, the set of states of $\mathcal{B}$ can be viewed as n copies (layers) of the Cartesian product of the sets of states $Q_1 \times \ldots \times Q_n$.

Any LTL formula ϕ over Π can be translated into a Büchi automaton $\mathcal{B}_\phi$ with alphabet 2^Π, such that $L(\phi) = L(\mathcal{B}_\phi)$. A number of standard translation algorithms (see, *e.g.*, [10], [12]) rely on a three-step procedure: First, the formula is normalized, second, it is translated into a generalized Büchi automaton and third, the obtained GBA is finally translated into a language-equivalent Büchi automaton (see Def. 4).

A weighted ω-automaton $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, Acc, \mathcal{W})$ is an ω-automaton, where $Q, q_{init}, \Sigma, \delta, Acc$ are defined in the usual way, and $\mathcal{W} : \delta \to \mathbb{N}$ is a function assigning a weight to each transition.

Let $\rho = q_0 q_1 \ldots$ and $\rho_{\text{fin}} = q_0 \ldots q_{l+1}$ be an accepting run over $w = w(0)w(1)\ldots$ and a finite run over $w_{\text{fin}} = w(0)\ldots w(l)$ of a weighted Büchi automaton $\mathcal{B}$, respectively. We use $\mathsf{Frag}(\rho) = \{q_i \ldots q_k \mid q_i, q_k \in F$ and $q_j \notin F$ for all $i < j < k\}$ and $\mathsf{Frag}(\rho_{\text{fin}}) = \{q_i \ldots q_k \mid q_i, q_k \in F, 0 \leq i \leq k \leq l$ and $q_j \notin F$, for all $i < j < k\}$ to denote the set of all finite fragments of ρ and ρ_{fin} that begin and end in an accepting state and do not contain any other accepting state. Note that each accepting run ρ and each finite run ρ_{fin} corresponds to a unique sequence of fragments. With a slight abuse of notation, we use

$$\mathcal{W}(q_i \ldots q_k) = \sum_{j=i}^{k-1} \mathcal{W}((q_j, w(j), q_{j+1}))$$

to denote the sum of the weights between the states of fragment $q_i \ldots q_k$ of a run ρ over w (or a finite run ρ_{fin} over w_{fin}).

B. Automata-Based Model Checking and Strategy Synthesis

Given a transition system $\mathcal{T}$ and a Büchi automaton $\mathcal{B}$, the *model checking problem* is to prove or disprove that all traces of $\mathcal{T}$ satisfy $\mathcal{B}$, whereas the *control strategy synthesis problem* is to find a trace of $\mathcal{T}$ that satisfies $\mathcal{B}$. Both of these problems can be addressed by constructing a product automaton $\mathcal{P}$ that captures all the behaviors of $\mathcal{T}$ satisfying $\mathcal{B}$ and searching for an accepting run of $\mathcal{P}$.

Definition 6 (Product Automaton) *A product automaton of a transition system $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$ and a BA $\mathcal{B} = (Q, q_{init}, \Sigma, \delta, F)$ is a Büchi automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B} = (Q_\mathcal{P}, q_{init,\mathcal{P}}, \delta_\mathcal{P}, F_\mathcal{P})$, where $Q_\mathcal{P} = S \times Q$; $q_{init,\mathcal{P}} = (s_{init}, q_{init})$; $F_\mathcal{P} = S \times F$; and $((s, q), (s', q')) \in \delta_\mathcal{P}$ if*

- *$(s, s') \in \mathcal{R}$ and $(q, \mathcal{L}(s), q') \in \delta$*

If $\mathcal{B} = (Q, q_{init}, \Sigma, \delta, F, \mathcal{W})$ is a weighted Büchi automaton, $\mathcal{P}$ is also weighted: $\mathcal{P} = (Q_\mathcal{P}, q_{init,\mathcal{P}}, \delta_\mathcal{P}, F_\mathcal{P}, \mathcal{W}_\mathcal{P})$, where $\mathcal{W}_\mathcal{P}\big(((s, q), (s', q'))\big) = \mathcal{W}\big((q, \mathcal{L}(s), q')\big)$, for all $((s, q), (s', q')) \in \delta_\mathcal{P}$.

The product automaton has a trivial alphabet, which is therefore omitted. An accepting run ρ of the product automaton projects onto a trace τ of $\mathcal{T}$ (denoted by $\tau = \alpha(\rho)$) that satisfies the property captured by the Büchi automaton $\mathcal{B}$. Vice versa, any trace of $\mathcal{T}$ satisfying the property corresponds to an accepting run of the product automaton. Furthermore, if there exists an accepting run $\rho_\mathcal{P}$ of $\mathcal{P}$, then there exists an accepting run $\rho'_\mathcal{P}$ of $\mathcal{P}$ in a *prefix-suffix structure, i.e.*, $\rho'_\mathcal{P} = \rho_{\text{pref}} \cdot (\rho_{\text{suf}})^\omega$ for some finite sequences ρ_{pref} and ρ_{suf} of states of $\mathcal{P}$, such that the first state of ρ_{suf} is an accepting state from $F_\mathcal{P}$.

The (weighted) product automaton can be viewed as a (weighted) graph (V, E) with the set of vertices V equal to the set of states Q_P and the set of edges E (and their weights) given by the transition function $\delta_\mathcal{P}$ (and the weight function $\mathcal{W}_\mathcal{P}$) in the expected way. A *simple path* in $\mathcal{P}$ is a sequence of states $p_i \ldots p_l$ such that $(p_j, p_{j+1}) \in \delta_\mathcal{P}$, for all $i \leq j < l$, and $p_j = p_{j'} \Rightarrow j = j'$, for all $i \leq j, j' \leq l$. A *cycle* is a sequence of states $p_i \ldots p_l p_{l+1}$, where $p_i \ldots p_l$ is a simple path and $p_{l+1} = p_i$. A state p' *reachable* from p if there is a simple path from p to p'.

Definition 7 (Maximal simple distance) The maximal simple distance *from $p_f \in F_\mathcal{P}$ to p in a weighted product automaton $\mathcal{P}$ is the maximal sum of edge weights on a simple path $p_i \ldots p_l$ from $p_i = p_f$ to $p_l = p$, such that $p_j \notin F_\mathcal{P}$, for all $i < j < l$.*

Efficient graph search algorithms can be used for finding a prefix ρ_{pref} (a simple path from the initial state to an accepting state in the product graph) followed by a periodically repeated suffix ρ_{suf} (a cycle in the product graph containing an accepting state) of an accepting run $\rho = \rho_{\text{pref}} \cdot (\rho_{\text{suf}})^{\omega}$ (a lasso-shaped path in the product graph). One of the standard algorithms to do so is *nested depth-first search (DFS)* [2], successfully implemented, for instance, in the pioneer model checker SPIN [14]. The (worst-case) running time complexity of this algorithm is linear in time and space with respect to the size (the number of states and transitions) of the product automaton $\mathcal{P}$.

III. Problem Formulation and Approach

Let us consider a robot moving in a partitioned environment with its motion capabilities modeled as a labeled transition system $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$ from Def. 1. Each region of the environment is modeled as a state of the transition system and the robot's ability to move between two regions is represented as transition between the corresponding states. In case several controlled robots are placed in the environment, the states of the transition systems encode positions of all the robots in the environmental regions *i.e.,* for k robots, a state corresponds to an k-tuple of regions, where the i-th element of the tuple is the region in which the i-th robot is placed. The transitions between the states reflect the simultaneous motion capabilities of all the robots. The labeling function $\mathcal{L}$ maps each state of the transition system to a subset of atomic propositions from Π that hold true in this state, such as "Vehicle x is in a safe region.".

There is a set of high-level missions to be accomplished by the robotic system expressed as a set of LTL formulas $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_n\}$ over Π with priorities of their satisfaction determined by a reward function $rew : \mathbf{\Phi} \to \mathbb{N}$. The value $rew(\phi_i)$ represents the reward that is gained if specification ϕ_i is accomplished. Without loss of generality, from now on, we assume that $rew(\phi_i) \geq rew(\phi_j)$, for all $1 \leq i \leq j \leq n$.

Given a trace τ of the transition system $\mathcal{T}$, we define *trace reward* as the sum of the rewards of all formulas from $\mathbf{\Phi}$ that are satisfied on this run.

Definition 8 (Trace Reward) Reward of a trace τ *of* $\mathcal{T}$ *is*

$$Rew(\tau) = \sum_{\{\phi_i | \tau \models \phi_i\}} rew(\phi_i). \tag{1}$$

We are now ready to formally state our problem of finding "the best" trace of $\mathcal{T}$, *i.e.,* "the least violating" motion of the robot (or the robots) in the environment with respect to the given set of mission specifications.

Problem 1 Given

- *a transition system* $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$*;*
- *a set of LTL formulas* $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_n\}$ *over* Π*; and*
- *a reward function* $rew : \mathbf{\Phi} \to \mathbb{N}$*,*

find *a trace* τ *of* $\mathcal{T}$ *that maximizes* $Rew(\tau)$ *from Eq. 1.*

Remark 1 *Note, that if* $rew(\phi_i) = 2^{n-i}$*, for each formula* $\phi_i \in \mathbf{\Phi}$*, then the set* $\mathbf{\Phi}$ *is in fact ordered according to the standard lexicographic ordering. In other words, it is always more important to satisfy* ϕ_i *than* $\phi_{i+1} \wedge \ldots \wedge \phi_n$*.*

A straightforward solution to Prob. 1 is to consider all the possible subsets $\mathbf{\Phi}_I = \{\phi_i \mid i \in I\}$, $I \subseteq \{1, \ldots, n\}$ of formulas from $\mathbf{\Phi}$ and to find a trace τ_I of $\mathcal{T}$ satisfying $\mathbf{\Phi}_I$ if such a trace exists. The search can be done using one of the known model-checking algorithms (*e.g.,* the automata-based algorithm from Sec. II). A trace τ_I maximizing $Rew(\tau_I)$ among the found ones maps to the desired robot path. However, this brute-force solution is not efficient as it requires up to 2^n model-checking procedure runs in the worst case.

In this paper, we suggest a method to alleviate the high computational demand of this straightforward solution. The main idea builds on the automata-based approach to model-checking. We construct a single weighted Büchi automaton $\mathcal{B}_{all}$ for formula $\bigwedge_{\phi_i \in \mathbf{\Phi}} \phi_i$ and capture the rewards of the LTL formulas through its weights. Then, a weighted product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}_{all}$ is built and an optimal accepting run of $\mathcal{P}$ is sought using a modification of the nested-DFS algorithm, with the computational complexity only slightly worse in comparison to the original nested-DFS. Roughly speaking, instead of up to 2^n model-checking procedure runs, we perform only a single execution of an altered model-checking algorithm.

IV. Problem Solution

This section introduces our solution to Prob. 1 in detail. First, we present the construction of the weighted Büchi automaton $\mathcal{B}_{all}$ and the weighted product automaton $\mathcal{P}$. Second, the modified nested-DFS is given. Third, we discuss the solution correctness, completeness and complexity.

A. Construction of the Weighted Automata

Consider the set of mission specifications $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_n\}$ that are translated (*e.g.,* using the algorithm from [10]) into generalized Büchi automata

$$\mathcal{G}_{\phi_1} = (Q_1, q_{init,1}, \Sigma, \delta_1, \mathcal{F}_1 = \{F_1^1, \ldots, F_1^{m_1}\}), \ldots$$
$$\ldots, \mathcal{G}_{\phi_n} = (Q_n, q_{init,n}, \Sigma, \delta_n, \mathcal{F}_n = \{F_n^1, \ldots, F_n^{m_n}\}),$$

respectively. Without loss of generality, we assume that $\mathcal{G}_{\phi_1}, \ldots, \mathcal{G}_{\phi_n}$ are all non-blocking. We build the weighted Büchi automaton $\mathcal{B}_{all}$ leveraging ideas from translation of generalized Büchi automata to Büchi automata (Def. 4) and from construction of a Büchi automaton for language intersection of several Büchi automata (Def. 5).

Definition 9 (Weighted Büchi automaton) *A weighted Büchi automaton* $\mathcal{B}_{all} = (Q, q_{init}, \Sigma, \delta, F, \mathcal{W})$ *is defined as follows:*

- $Q = Q_1 \times \ldots \times Q_n \times (\{(j, l) \mid 1 \leq j \leq n, 1 \leq l \leq m_i\} \cup \{(0, 0)\})$*;*
- $q_{init} = (q_{init,1}, \ldots, q_{init,n}, (0, 0))$*;*
- $t = \big((q_1, \ldots, q_n, (j, l)), \sigma, (q_1', \ldots, q_n', (j', l'))\big) \in \delta$ *if* $(q_i, \sigma, q_i') \in \delta_i$*, for all* $i \in \{1, \ldots, n\}$*, and*

1) $(j,l) = (0,0)$ *and*
   a) $(j',l') = (0,0)$. *Then* $\mathcal{W}(t) = 0$.
   b) $j' > 0, l' = 1$. *Then* $\mathcal{W}(t) = rew(\phi_{j'})$.
2) $j \neq 0$ *and*
   a) $(j',l') = (j,l)$ *and* $q_j \notin F_j^l$. *Then* $\mathcal{W}(t) = 0$.
   b) $l \neq m_j, (j',l') = (j,l+1)$ *and* $q_j \in F_j^l$. *Then* $\mathcal{W}(t) = 0$.
   c) $l = m_j, j < n$, $j \leq j'$, $l' = 1$ *and* $q_j \in F_j^l$. *Then* $\mathcal{W}(t) = rew(\phi_{j'})$.
   d) $l = m_j, (j',l') = (0,0)$, *and* $q_j \in F_j^l$. *Then* $\mathcal{W}(t) = 0$.

- $F = Q_1 \times Q_2 \times \ldots \times Q_n \times \{(0,0)\}$.

Loosely speaking, the set of states of the automaton $\mathcal{B}_{all}$ can be viewed as *layers*, where the j-th layer consists of m_j *components*, for all $1 \leq j \leq n$. Each component then involves a copy of each element from the Cartesian product $Q_1 \times \ldots \times Q_n$. Within the j-th layer, the l-th component is connected to the $(l+1)$-th component through transitions leading from F_j^l. The j-th layer is connected to the j'-th through transitions leading from $F_j^{m_j}$, for all $j+1 \leq j' \leq n$. These transitions are labeled with the reward $rew(\phi_{j'})$. Besides that, the layer 0 consist only one component $(0,0)$, whose states are all and the only ones accepting. From this component, transition leads to the first component of each layer, and dually, from the last component of each layer, transitions lead to this component.

Note that the automaton $\mathcal{B}_{all}$ accepts all words satisfying specifications $\bigwedge_{\phi_i \in \mathbf{\Phi}_I} \phi_i$, for all $\mathbf{\Phi}_I \subseteq \mathbf{\Phi}$. The weights associated with transitions connecting the layers determine the "quality" of a particular run, *i.e.*, they capture which formulas are satisfied by this run. Particularly, if an accepting run enters the j-th layer infinitely many times, then it intersects all $F_j^l \in \mathcal{F}_j$ infinitely many times and thus the satisfaction of ϕ_j is guaranteed. Furthermore, such a run contains infinitely many transitions weighted with $rew(\phi_j)$.

Formally, the purpose of the weights of $\mathcal{B}_{all}$ is summarized as follows. Let us denote the component of a state as $component(q_1, \ldots, q_n, (j,l)) = (j,l)$.

Definition 10 (Run Reward) *The reward of a run ρ of $\mathcal{B}_{all}$ is*

$$Rew(\rho) = \max\big\{C \mid C = \mathcal{W}(q_i \ldots q_l) \text{ for infinitely many fragments } q_i \ldots q_l \in \mathsf{Frag}(\rho)\big\}.$$

Intuitively, a run ρ can be split into a sequence of fragments that is associated with a respective sequence of fragment weights. The run reward is equal to the maximal weight that appears in the sequence of fragment weights infinitely many times.

Lemma 1 *Consider a word $w = w(0)w(1)\ldots$, where $w \models \mathbf{\Phi}_I$ and $w \not\models \phi$, for all $\phi \notin \mathbf{\Phi}_I$. There exists an accepting run $\rho = q_0 q_1 \ldots$ of $\mathcal{B}_{all}$ over w, such that $Rew(\rho) = \sum_{\phi_i \in \mathbf{\Phi}_I} rew(\phi_i)$. Furthermore, for each accepting run $\rho' = q'_0 q'_1 \ldots$ of $\mathcal{B}_{all}$ over w it holds, that $Rew(\rho') \leq \sum_{\phi_i \in \mathbf{\Phi}_I} rew(\phi_i)$.*

Proof: If $w \models \mathbf{\Phi}_I$ then there is an accepting run $\rho_i = q_0 q_1 \ldots$, for all $\phi_i \in \mathbf{\Phi}_I$. Let $I = \{i_1, \ldots, i_j\}$. According to the construction of $\mathcal{B}_{all}$ (Def. 9), there exists a run $\rho = p_0 p_1 \ldots$ of $\mathcal{B}_{all}$, such that each fragment $p_k \ldots p_{k'} \in \mathsf{Frag}(\rho)$ satisfies the following.

$$\begin{aligned}
component(p_{l_1}) &= (0,0)\\
component(p_{(l_1+1)}) &= \ldots = component(p_{l_2}) = (i_1, 1)\\
&\ldots\\
component(p_{l_3}) &= \ldots = component(p_{l_4}) = (i_1, |\mathcal{F}_{i_1}|)\\
component(p_{(l_4+1)}) &= \ldots = component(p_{l_5}) = (i_2, 1)\\
&\ldots\\
component(p_{l_6}) &= \ldots = component(p_{l_7}) = (i_2, |\mathcal{F}_{i_2}|)\\
&\ldots\\
component(p_{(l_8)}) &= \ldots = component(p_{l_9}) = (i_j, 1)\\
&\ldots\\
component(p_{l_{10}}) &= \ldots = component(p_{l_{11}}) = (i_j, |\mathcal{F}_{i_j}|)\\
component(p_{l_{(11+1)}}) &= (0,0)
\end{aligned}$$

where $p_{l_1} = p_k$, $p_{l_{(11+1)}} = p_{k'}$. The total weight of such a fragment and hence also the reward of ρ is equal to $\sum_{\phi_i \in \mathbf{\Phi}_I} \phi_i$ directly from the construction of $\mathcal{B}_{all}$.

On the other hand, assume that there exists a run $\rho' = q'_0 q'_1 \ldots$ of $\mathcal{B}_{all}$ over w such that $Rew(\rho') > \sum_{\phi_i \in \mathbf{\Phi}_I} rew(\phi_i)$. From the construction of the automaton $\mathcal{B}_{all}$, this means that there exist infinitely many fragments $p_k \ldots p_{k'} \in \mathsf{Frag}(\rho')$ with their weight larger than $\sum_{\phi_i \in \mathbf{\Phi}_I} rew(\phi_i)$. Therefore, there exist $\phi_l \notin \mathbf{\Phi}_I$, and states $p'_1, \ldots, p'_{|\mathcal{F}_l|}$ of $\mathcal{B}_{all}$, such that $component(p'_j) = (l,j)$, for all $j \in \{1, \ldots, |\mathcal{F}_l|\}$. Thus, the run ρ' can be projected to an accepting run of $\mathcal{B}_l$ over w, which is in contradiction with our assumption that $w \not\models \phi_l$ for all $\phi_l \notin \mathbf{\Phi}_I$. ∎

The second step of our algorithm is the construction of a product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}_{all} = (Q_\mathcal{P}, p_{init}, \delta_\mathcal{P}, F_\mathcal{P}, \mathcal{W}_\mathcal{P})$ (see Def. 6). Based on Lemma 1, the product automaton satisfies the following:

Lemma 2 *Let τ be a trace of $\mathcal{T}$. Then, there exists a run $\rho_\mathcal{P}$ of $\mathcal{P}$ with $\tau = \alpha(\rho_\mathcal{P})$ such that the reward $Rew(\tau) = Rew(\rho_\mathcal{P})$. Moreover, $Rew(\tau) \geq Rew(\rho'_\mathcal{P})$ for all $\rho'_\mathcal{P}$ with $\alpha(\rho'_\mathcal{P}) = \tau$.*

Proof: The proof follows directly from Lemma 1 and the fact that for each trace τ that produces a word $w = w(0)w(1)\ldots$ accepted by a run $\rho = q_0 q_1 \ldots$ of $\mathcal{B}_{all}$, there exists an accepting run $\rho_\mathcal{P} = p_0 p_1 \ldots$ in $\mathcal{P}$, such that $\mathcal{W}((p_i, p_{i+1})) = \mathcal{W}((q_i, w(i), q_{i+1}))$ and $q_i \in F \iff p_i \in F_\mathcal{P}$, for all $i \geq 0$. ∎

Lemma 3 *For each run $\rho_\mathcal{P}$ there exists a run $\rho'_\mathcal{P}$ in prefix-suffix structure, such that $Rew(\rho_\mathcal{P}) = Rew(\rho'_\mathcal{P})$.*

Proof: Because $\rho_{\mathcal{P}} = p_0 p_1 \dots$ is infinite, there exist a state $p \in F_{\mathcal{P}}$ that appears on $\rho_{\mathcal{P}}$ infinitely many times and there exist a fragment $p \dots p'$ starting in p such that $\mathcal{W}(p \dots p') = Rew(\rho_{\mathcal{P}})$. Because p occurs on $\rho_{\mathcal{P}}$ infinitely many times, p is reachable from p'. Therefore, run $\rho_{\mathcal{P}}$ is a sequence of states $\rho_{\mathcal{P}} = p_0 p_1 \dots p \dots p' \dots p \dots$. Let $\rho'_{\mathcal{P}} = p_0 p_1 \dots (p \dots p' \dots p)^{\omega}$. Run $\rho'_{\mathcal{P}}$ is in prefix-suffix structure, it is accepting and $Rew(\rho'_{\mathcal{P}}) = Rew_(\rho_{\mathcal{P}})$. ■

The three lemmas above provide us with guidance on computing the trace of $\mathcal{T}$ with the maximal reward: it is enough to compute a run of $\mathcal{P}$, in the prefix-suffix structure, that maximizes $Rew(\rho_{\mathcal{P}})$ and project this run into a trace of $\mathcal{T}$. This is stated in the following proposition.

Proposition 1 *Let $\tau = s_0 s_1 \dots$ be a trace of $\mathcal{T}$, such that $\tau \models \mathbf{\Phi}_I$ and $\tau \not\models \phi$, for all $\phi \notin \mathbf{\Phi}_I$. Then, there exists an accepting run $\rho_{\mathcal{P}} = (s_0, q_0)(s_1, q_1) \dots$ in $\mathcal{P}$ such that*

(i) *$\rho_{\mathcal{P}}$ is in prefix-suffix structure and*
(ii) *$Rew(\rho_{\mathcal{P}}) = \sum_{\phi_i \in \mathbf{\Phi}_I} \phi_i$.*

The remaining task is to find a run $\rho_{\mathcal{P}}$ satisfying the condition (i) of Proposition 1 and maximizing $Rew(\rho_{\mathcal{P}})$. The problem thus reduces to searching for a reachable cycle c (a repeated run suffix) in $\mathcal{P}$ beginning (and thus also ending) in an accepting state that maximizes the value

$$Rew(c) = \max_{p_i \dots p_l \in \mathsf{Frag}(c)} \mathcal{W}(p_i \dots p_l) \tag{2}$$

among all such cycles. The following lemma helps narrow down the search even further, showing that it is enough to search for a particular type of cycle.

Lemma 4 *Given a cycle c in $\mathcal{P}$ and a fragment $p_i \dots p_l \in \mathsf{Frag}(c)$, there exists a simple path $p_i \dots p_l$, such that $\mathcal{W}(p_i \dots p_l) = 0$.*

Proof: From the construction of the automaton $\mathcal{B}_{all}$, it follows that if there is a simple path from $(q_1, \dots, q_n, (0,0)) \in Q$ to $(q'_1, \dots, q'_n, (i,j)) \in Q$ in the automaton $\mathcal{B}_{all}$, then there exists a simple path from $(q_1, \dots, q_n, (0,0))$ to $(q'_1, \dots, q'_n, (0,0))$ that contains only states $\mathbf{q} \in Q$, such that $component(\mathbf{q}) = (0,0)$. Thus, if there is a simple path from $(s, q_1, \dots, q_n, (0,0)) \in Q_{\mathcal{P}}$ to $(s', q'_1, \dots, q'_n, (i,j)) \in Q_{\mathcal{P}}$ in the product automaton $\mathcal{P}$, then there exists also a simple path from $(s, q_1, \dots, q_n, (0,0))$ to $(s', q'_1, \dots, q'_n, (0,0))$ that contains only states $p \in Q_{\mathcal{P}}$, such that $component(p) = (0,0)$ The reward of such a simple path is 0. ■

Thanks to Lemma 4, it is enough to search for a cycle c maximizing Eq. 2, such that $\mathcal{W}(p_i \dots p_l) = 1$, for all fragments $p_i \dots p_l \in \mathsf{Frag}(c)$, but one. Hence, without loss of generality, we can consider only cycles $c = p_i \dots p_l p_{l+1} \dots p_i$ such that $\mathcal{W}(p_i \dots p_l) \neq 0$ only for the first fragment $p_i \dots p_l$ of the cycle. Such a cycle can be found by adapting standard nested depth-first search algorithm as we will show in the following section.

Proposition 2 *A maximal-reward trace of $\mathcal{T}$ can be obtained as a projection $\alpha(p_0 \dots p_l) \cdot (\alpha(c))^{\omega}$ of a path $p_0 \dots p_l$ and a cycle $c = p_{l+1} \dots p_i p_{i+1} \dots p_{l+1}$, such that*

- *$p_0 = q_{init,\mathcal{P}}$, $(p_l, p_{l+1}) \in \delta$, $p_{l+1} \in F_{\mathcal{P}}$,*
- *$\mathcal{W}(p_{l+1} \dots p_i)$ for the first fragment $p_{l+1} \dots p_i \in \mathsf{Frag}(c)$ of the cycle is maximized, and*
- *$\mathcal{W}(p_j \dots p_k) = 0$, for all fragments $p_j \dots p_k \in \mathsf{Frag}(p_{i+1} \dots p_{l+1})$.*

B. Weighted Nested Depth-First Search

This section aims at search for a path $p_0 \dots p_l$ followed by a cycle $p_{l+1} \dots p_{l+1}$ satisfying conditions of Prop. 2. The solution is summarized in Alg. 1 to Alg. 3. The external functions used in the algorithms are summarized and explained in Table I.

First, let us focus on a solution to the following subproblem: Given an accepting state $p_f \in F_{\mathcal{P}}$, find a cycle $c = p_f \dots p_i p_{i+1} \dots p_f$ that maximizes value $\mathcal{W}(p_f \dots p_i)$ in Eq. 2 for the first fragment $p_f \dots p_i \in \mathsf{Frag}(c)$ among all cycles that begin and end in p_f. A modification of breath-first graph search algorithm as described in Alg. 3 can be used to do so in $\mathcal{O}(|\mathcal{P}|)$ time and space thanks to the fact that the individual layers connected through non-zero weighted transitions form a directed acyclic graph. Intuitively, the algorithm systematically searches the graph $\mathcal{P}$ and maintains for each state p the approximation of the maximal simple distance (Def. 7) from p_f to p. The correctness of the algorithm relies on the fact, that when p is processed on line 2 of the procedure propagate (Alg. 4), the value of $p.dist$ is set to the actual maximal simple distance from p_f to p. When all states that are reachable from state p_f are visited in Alg. 4, the second phase (lines 14-25) of Alg. 3 is executed to check whether p_f is also reachable from p, considering p one by one in descending order of their $p.dist$.

Second, the cycle satisfying conditions of Prop. 2 can be found by running Alg. 3 from each $p_f \in F$ reachable from the initial state, potentially traversing the whole graph $|F|$-times. However, leveraging ideas from nested DFS algorithm, the complexity can be reduced. The idea is to run Alg. 3 from states in $F_{\mathcal{P}}$ in particular order that ensures the states visited during previous executions of Alg. 3 do not need to be visited again. In particular, in the standard nested DFS it holds that if a cycle is being sought from a state p'_f (so-called inner-search) that is reachable from p_f and the search is unsuccessful, then later, when a cycle is sought from p_f, the states visited in the inner-search from p'_f do not have to be considered again. Based on this idea, we formulate the following lemma that explains the correctness of our approach.

Lemma 5 *Let $p'_f \in F_{\mathcal{P}}$ be reachable from $p_f \in F_{\mathcal{P}}$ and $p \in Q_{\mathcal{P}}$ be reachable from both p_f and p'_f. If there exists a cycle c from state $p_f \in F_{\mathcal{P}}$ containing state p, then there exists a cycle c' from $p'_f \in F_{\mathcal{P}}$ with reward $Rew(c') \geq Rew(c)$.*

Proof: Because p_f is reachable from p, p is reachable from p'_f, and p'_f is reachable from p_f, then p_f is

reachable from p'_f. Therefore, there exists a cycle $c' = p'_f \dots p_f \dots p \dots p_f \dots p'_f$, where $p_f \dots p \dots p_f = c$. Clearly $Rew(c') \geq Rew(c)$. ∎

find_arbitrary_trace($\mathcal{T}$)	returns and arbitrary trace of TS $\mathcal{T}$
find_path($\mathcal{P}, p, p_f$)	returns a path from p to p_f in $\mathcal{P}$
successors(p)	returns the immediate successors of p in $\mathcal{P}$
$stack$.push(p)	inserts p on the top of $stack$
$stack$.top()	reads from the top of $stack$
$stack$.top_and_pop()	destructively reads from the top of $stack$
$stack$.pop()	removes element from the top of $stack$
reverse($stack$)	returns the elements of $stack$ in the reversed order

TABLE I: List of functions used in Alg. 1–4

Alg. 1 weighted_nested_DFS($\mathcal{P}$)

Input: product automaton $\mathcal{P}$
Output: solution to Prob. 1

$weight\_max = 0; prefix\_max = \epsilon; cycle\_max = \epsilon$
$stack\_outer = \text{empty}; visited\_outer = \varnothing$
$visited\_inner = \varnothing; visited\_ps = \varnothing$
for all $p \in Q_{\mathcal{P}}$ **do**
  $p.dist = 0; p.pred = \perp$
end for
$run :=$ DFS($\mathcal{P}, p_{init}$)
if $run \neq \epsilon$ **then**
  return $trace := \alpha(run)$
else
  return find_arbitrary_trace($\mathcal{T}$)
end if

Alg. 2 DFS($\mathcal{P}, p$)

Input: product automaton $\mathcal{P}$, state p
Output: run of $\mathcal{P}$ satisfying conditions of Prop. 2.

$stack\_outer$.push(p); $visited\_outer := visited\_outer \cup \{p\}$
repeat
  $p' := stack\_outer$.top()
  if successors(p') $\setminus$ $visited\_outer \neq \varnothing$ **then**
    pick $p'' \in$ successors(p') $\setminus$ $visited\_outer$
    $stack\_outer$.push(p'')
    $visited\_outer := visited\_outer \cup \{p''\}$
  else
    $stack\_outer$.pop()
    if $p' \in F_{\mathcal{P}}$ **then**
      $p'.dist := 0$; $p'.pred = \perp$
      $cycle :=$ longest_cycle_search($\mathcal{P}, p'$)
      if $p'.dist > weight\_max$ **then**
        $weight\_max := p'.dist; cycle\_max := cycle$
        $prefix\_max :=$ reverse($stack\_outer$)
      end if
    end if
  end if
until ($stack\_outer = \text{empty} \vee weight\_max = n$)
return $prefix\_max \cdot (cycle\_max)^{\omega}$

C. Algorithm Summary and Analysis

The overall solution can be summarized as follows:

1) Each of the formulas $\phi \in \mathbf{\Phi}$ is translated into a generalized Büchi automaton $\mathcal{G}_\phi$
2) A weighted Büchi automaton $\mathcal{B}_{all}$ is built (see Def. 9)
3) A weighted product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}_{all}$ is constructed (see Def. 6).
4) Alg. 1 is run on $\mathcal{P}$.

Alg. 3 longest_cycle_search($\mathcal{P}, p_f$)

Input: product automaton $\mathcal{P}$, accepting state $p_f \in F_{\mathcal{P}}$
Output: cycle $(p_f, \dots, p_f)$ maximizing Eq. 2 (if one exists)

$queue\_curr := (p_f)$
for all $1 \leq i \leq n$ **do**
  $queues\_all[i] := \text{empty}$
end for
$to\_search\_from := \varnothing$
propagate($\mathcal{P}, queue\_curr, queues\_all, search\_from$)
for all $1 \leq i \leq n$ **do**
  $queue\_curr := queues\_all[i]$
  if $queue\_current \neq \text{empty}$ **then**
    propagate($\mathcal{P}, queue\_curr, queues\_all, search\_from$)
  end if
end for
$cycle := \epsilon$
order $search\_from$ decreasingly according to $p.dist$
while $search\_from \neq \text{empty}$ **do**
  $p := search\_from$.top_and_pop()
  $path\_suf :=$ find_path($\mathcal{P}, p, p_f$)
  if $path \neq \epsilon$ **then**
    $p_f.dist := p.dist; path\_pref := \epsilon$
    repeat
      $path\_pref := (p.pred) \cdot (path\_pref); p := p.pred$
    until $p = p_f$
    return $cycle := (path\_pref) \cdot (path\_suf)$
  end if
end while
return $cycle := \epsilon$

Alg. 4 propagate($\mathcal{P}, queue\_curr, queues\_all, search\_from$)

repeat
  $p := queue\_curr$.front_and_pop()
  if $p \notin visited\_inner$ **then**
    $visited\_inner := visited\_inner \cup \{p\}$
    for all $p' \in$ succs(p) **do**
      if $p'.dist < p.dist + \mathcal{W}_{\mathcal{P}}(p, p')$ **then**
        $p'.dist := p.dist + \mathcal{W}_{\mathcal{P}}(p, p')$; $p'.pred := p$
        if $component(p') = (0,0) \wedge p' \notin search\_from$ **then**
          $search\_from = search\_from \cup \{p'\}$
        end if
        if $component(p') = component(p)$ **then**
          $queue\_curr$.push(p')
        else if $component(p') = i$ for some $i \geq 1$ **then**
          $queues\_all[i]$.push(p')
        end if
      end if
    end for
  end if
until $queue\_curr = \text{empty}$

Correctness and Correctness: Based on Lemmas 1–5 and Propositions 1–2, the soundness and completeness properties of the algorithm are summarized in the following theorem.

Theorem 1 (Soundness and completeness) *Given a transition system $\mathcal{T}$, a set of LTL formulas $\mathbf{\Phi}$ and the reward*

function rew, the suggested algorithm returns the solution to Prob. 1.

Theorem 2 *Let $|\mathcal{T}|$ and $|\mathbf{\Phi}|$ denote the size of the input transition system and the size of the missions specification, respectively. The computational complexity of Alg. 1 is in $\mathcal{O}(|\mathcal{P}| \cdot \log|\mathcal{P}|)$, where $|\mathcal{P}|$ is the size of the product automaton, which is in $\mathcal{O}(|\mathcal{T}| \cdot 2^{\mathcal{O}(|\mathbf{\Phi}|)})$.*

Discussion: The translation from an LTL formula ϕ into a generalized Büchi automaton can be done in in $2^{\mathcal{O}(|\phi|)}$ time and space. In particular, one of the well-known translation algorithms [10] transforms ϕ into a generalized Büchi automaton with at most $2^{|\phi|}$ states and $|\phi|$ sets in its acceptance condition. If the obtained GBAs for specifications $\phi_1 \dots \phi_n \in \mathbf{\Phi}$ are all non-blocking, the worst-case size of $\mathcal{B}_{all}$ is $2^{(|\mathbf{\Phi}|)} \cdot (1 + |\mathbf{\Phi}|)$. On the other hand, in case k of the obtained GBAs are blocking, the worst-case size of $\mathcal{B}_{all}$ is $2^{(|\mathbf{\Phi}|+k)} \cdot (|\mathbf{\Phi}| + k + 1)$. Although the size of the resulting GBA is exponential with respect to the size of the input specification, the sizes of the individual formulas are usually small and in many cases, the GBAs are significantly smaller than the worst-case bound. Many optimizations techniques have been also developed among the formal methods literature to reduce the sizes of the GBAs.

The size of the product automaton $\mathcal{P}$ is $|\mathcal{T}| \cdot |\mathcal{B}_{all}|$ in the worst case, with at most $|\mathcal{T}| \cdot 2^{(|\mathbf{\Phi}|+k)}$ in one component, where k is the number of blocking GBAs obtained in translation of the formulas from $\mathbf{\Phi}$. The cumulative number of steps made in sorting the set $search\_from$ on line 14 of Alg. 3 is bounded by $\mathcal{O}(|L_\mathcal{P}| \cdot \log|L_\mathcal{P}|)$, where $|L_\mathcal{P}| = \{p \in Q_\mathcal{P} \mid component(p) = (0,0)\}$ is the size of the initial component of $\mathcal{P}$. Altogether, the complexity of Alg. 1 is in $\mathcal{O}(|\mathcal{P}| + |L_\mathcal{P}| \cdot \log|L_\mathcal{P}|)$.

In contrast, the "brute-force" approach that tries to find a trace satisfying $\mathbf{\Phi}_I$, for each $\mathbf{\Phi}_I \subseteq \mathbf{\Phi}$ has the worst time complexity characterized as follows. A Büchi automaton $\mathcal{B}_{\mathbf{\Phi}_I}$ for $\mathbf{\Phi}_I$ can be constructed with $2^{|\mathbf{\Phi}_I|} \cdot |\mathbf{\Phi}_I|$ number of states in the worst case. A nested DFS algorithm is then run on $\mathcal{P} = \mathcal{T} \cdot \mathcal{B}_{\mathbf{\Phi}_I}$, reaching complexity $\mathcal{O}(|\mathcal{P}|)$. Hence, the solution is linear with respect to the size of $|\mathcal{T}| \cdot \sum_{\mathbf{\Phi}_I \subseteq \mathbf{\Phi}} 2^{|\mathbf{\Phi}_I|} \cdot |\mathbf{\Phi}_I|$.

The benefit of our algorithm (Alg. 1) in comparison to the brute-force solution increases with the increasing number of *non-blocking* GBAs obtained from the translation from LTL formulas. Note, that for some LTL formulas, the smallest existing corresponding GBA is non-blocking. In particular many useful specifications, such as $\mathsf{F}\phi$ (reachability), $\mathsf{GF}\,\phi$ (surveillance), $\mathsf{GF}\,\phi_1 \Rightarrow \mathsf{GF}\,\phi_2$ (reactivity), $\mathsf{G}(\phi_1 \Rightarrow \mathsf{F}\phi_2)$ (response), or $\mathsf{F}(\phi_1 \wedge \mathsf{F}\phi_2)$ (sequencing), where ϕ, ϕ_1, ϕ_2 are arbitrary Boolean combinations of atomic propositions, belong to this class.

V. Rescue Mission Example

Let us consider an example of a complex military rescue mission. Assume that friendly units F_1, F_2, F_3 have been captured in an enemy territory. They are guarded by enemy units (called targets) $T_1, \dots, T_7$, which need to be engaged before an autonomous vehicle can proceed to pick up the captured friendly units and bring them to the friendly base. A particular configuration is depicted in Fig. 1.(a). While friendly units F_1 and F_2 can be rescued by engaging targets T_1, T_4, and T_2, T_6, T_7, respectively, unit F_3 can be rescued by engaging targets T_3 and T_2. Suppose that we have two unmanned aerial vehicles (UAVs) V_1 and V_2 and an autonomous ground vehicle V_3 under our command, with their capabilities and weaknesses as described below.

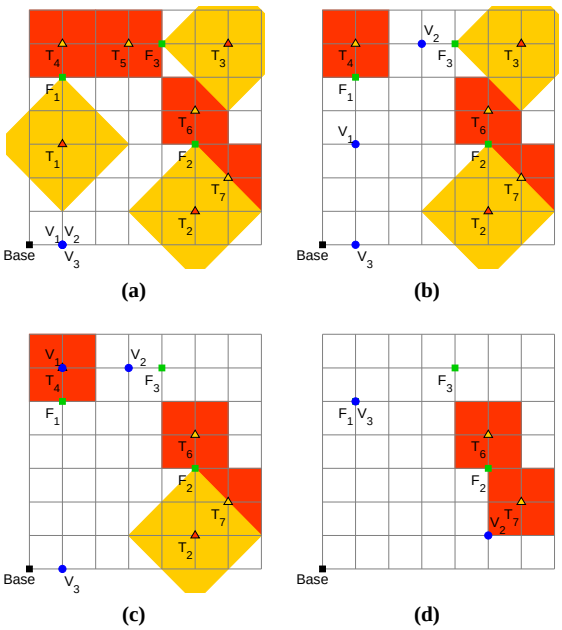

Fig. 1: An example of a mission featuring conflicting specifications. The captured friendly units F_1, F_2, and F_3 are shown as green squares and the enemy units (the targets) $T_1, \dots, T_7$ are illustrated as triangles. The respective firing ranges of the targets are depicted as yellow and red squares around the targets. The friendly vehicles V_1, V_2, V_3 are the blue dots, that can move along the edges of the rectangular grid. A visit of vehicle V_1 or V_2 into a location with a target is considered an engagement of the target. On the other hand, vehicle V_1, V_2, or V_3 entering a region within the firing range of a target to which it is vulnerable results in the loss of the vehicle. These rules are captured through irreversible transitions of the underlying state transition system.

- V_1 can engage T_1, T_3, is vulnerable to T_2, T_5, and can engage T_4, T_6, T_7 at the cost of self-destruction (*i.e.*, it can be sacrificed to engage a target T_4, T_6, or T_7).
- V_2 can engage T_2, T_5, is vulnerable to T_1, T_3, and can engage T_4, T_6, T_7 at the cost of self-destruction.
- V_3 can pickup and transport F_1, F_2, F_3, but is vulnerable to all active targets.

The mission is to rescue and pickup the friendly units F_1, F_2 and F_3 and bring them to the base ($Base$). At the same time, the goal is not to loose any of the vehicles V_1, V_2, V_3. Let $p_{V_i}^{F_j}$, $p_{V_k}^{Base}$ and a_{V_ℓ} denote the atomic propositions "Vehicle V_k is at the location of the friendly unit F_j", "Vehicle V_i is at $Base$", and "Vehicle V_ℓ is active", respectively. Individual goals are expressed as LTL formulas (see Table II) and assigned priorities through the reward function. The reward function, among others, specifies that

Mission Specification	**LTL Formula:** ϕ	$rew(\phi)$
Pickup F_i, and bring it to $Base$, for all $i \in \{1,2,3\}$	$\mathsf{F}(p_{V_3}^{F_i} \wedge \mathsf{F}(p_{V_3}^{Base}))$, for $i \in \{1,2,3\}$	10
Do not pick up F_3 before picking up F_i, for all $i \in \{1,2\}$	$\mathsf{G}(p_{V_3}^{F_3} \Rightarrow \mathsf{G}(\neg p_{V_3}^{F_i}))$, for $i \in \{1,2\}$	10
Do not lose vehicle V_k and bring V_k to $Base$, for all $k \in \{1,2,3\}$	$\mathsf{G}(\neg a_{V_k}) \wedge \mathsf{F}(p_{V_k}^{Base})$, for $k \in \{1,2,3\}$	1

TABLE II: Mission specification.

saving the friendly units is more important than not loosing the vehicles V_1, V_2, V_3. Note, that because enemy target T_7 cannot be destroyed by any of the vehicles at no cost to their integrity, at least one vehicle must be sacrificed to save the friendly units. Although not so obvious, one can also observe that friendly units F_1 and F_2 cannot both be rescued.

In order to validate our algorithm, we developed a C++ implementation which takes as an input a deterministic transition system and a list of generalized Büchi automata obtained from the LTL formulas with the use of an off-the-shelf tool such as LTL2BA [11]. The reward gained if the optimal control strategy of the vehicles is applied is 32 units, as expected. Figures 1.(b)–1.(d) illustrate different stages of the system run. First, vehicles V_1 and V_2 engage enemy targets T_1 and T_5, respectively (Fig. 1.(b)). Then, V_1 destroys enemy target T_3 before launching a self-destructive attack on T_4 (Fig. 1.(c)). Later, vehicle V_2 engages enemy target T_2, and vehicle V_3 proceeds to pickup F_1 and F_3, in that order (see Fig. 1.(d)). Finally, the remaining vehicles return to $Base$.

VI. Conclusion and Future Work

In this paper, we have studied the least-violating controller synthesis problem, *i.e.,* roughly speaking, to find a trajectory that satisfies the most important pieces of the specification, when the specification can not be satisfied as a whole. We have proposed an algorithm that provides substantial computational savings when compared to a straightforward solution. We have analyzed the proposed algorithm in terms of correctness, completeness and computational complexity. We have also demonstrated the performance of the proposed algorithm on an illustrative example.

There are many directions for future work. In particular, synthesis of *optimal* strategies that are least violating, and also synthesis of such strategies to be implemented in *dynamic environments* are possible directions for future work.

References

[1] Isaac Asimov. *I, Robot*. Gnome Press, 1950.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[3] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model repair for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 326–340. Springer-Verlag, 2011.

[4] Amit Bhatia, Lydia .E. Kavraki, and Moshe .Y. Vardi. Sampling-based motion planning with temporal goals. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2689–2696, 2010.

[5] Francesco Buccafurri, Thomas Eiter, Georg Gottlob, and Nicola Leone. Enhancing model checking in verification by AI techniques. *Artificial Intelligence*, 112(1-2):57 – 104, 1999.

[6] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In *Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 52–67, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] Costas Courcoubetis and Mihalis Yannakakis. Markov decision processes and regular events. In *IEEE Transactions on Automatic Control*, 1998.

[8] Werner Damm and Bernd Finkbeiner. Does it pay to extend the perimeter of a world model? In *Proceedings of the International Symposium on Formal Mehods (FM)*, pages 12–26, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] Georgios E. Fainekos. Revising temporal logic specifications for motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[10] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, pages 53–65, London, UK, UK, 2001. Springer-Verlag.

[11] Paul Gastin and Denis Oddoux. LTL2BA tool, viewed September 2012. URL: http://www.lsv.ens-cachan.fr/ gastin/ltl2ba/.

[12] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 3–18, London, UK, UK, 1996. Chapman & Hall, Ltd.

[13] Kris Hauser. The minimum constraint removal problem with three robotics applications. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

[14] Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[15] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 2222–2229, 2009.

[16] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning with deterministic μ-calculus specifications. In *Proceedings of the American Control Conference (ACC)*, 2012.

[17] Kangjin Kim, Georgios Fainekos, and Sriram Sankaranarayanan. On the revision problem of specification automata. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[18] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287 –297, 2008.

[19] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Automatic Control*, 25(6):1370–1381, 2009.

[20] Morteza Lahijanian, Joe Wasniewski, Sean .B. Andersson, and Calin Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3227 –3232, 2010.

[21] Vasumathi Raman and Hadas Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using ltlmop. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, pages 663–668, 2011.

[22] Vasumathi Raman and Hadas Kress-Gazit. Automated feedback for unachievable high-level robot behaviors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5156–5162, 2012.

[23] Stephen. L. Smith, Jana Tumova, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal logic Constraints. *International Journal of Robotics Research*, 30(14):1695–1708, 2011.

[24] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon temporal logic planning for dynamical systems. In *Proceedings of the IEEE Conference on Decision and Control and the Chinese Control Conference (CDC/CCC)*, pages 5997 –6004, 2009.