2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Shortcut Through an Evil Door: Optimality of Correct-by-Construction Controllers in Adversarial Environments

Gangyuan Jing, Rüdiger Ehlers and Hadas Kress-Gazit

Abstract— **A recent method to obtain correct robot controllers is to automatically synthesize them from high-level robot missions that are specified in temporal logic. In this context, we aim for controllers that are *optimal*, i.e., do not let the robot take unnecessarily costly paths to reach its goals. Previous work on obtaining optimal synthesized robot controllers either ignored interactions with the environment, or assumed a cooperative environment.**

In this paper, we solve the problem of obtaining optimal robot controllers for adversarial environments. Our main observation is that the quality of a path to a goal has two dimensions: (1) the number of phases in which the robot waits for the environment to perform some actions and (2) the cost of the robot's actions to reach the goal. Our synthesis algorithm can take any prioritization over the possible cost combinations into account, and computes the optimal strategy in a symbolic manner, despite the fact that the action costs can be non-integer. We show the scalability of the new algorithm by example of a delivery problem.

I. INTRODUCTION

As the capabilities of autonomous systems and robots are rapidly improving, methods for generating controllers for robots that fulfill various complex missions are of wide interest in many communities, such as planning, machine learning, and control. Current work [1]–[3], [7], [9]–[11], [13]–[15], [19] aims at generating provably correct controllers from high-level mission requirements using formal methods such as model checking and reactive synthesis. In these methods, controllers are usually generated based on the discrete abstraction of the problem and then implemented continuously. The correctness of the robot's behavior on the discrete abstraction gives rise to the same guarantee in the continuous sense. To specify the mission of the robot, temporal logic is widely used as it enables us to reason about the mission specification over time. We can formulate various complex robot mission properties, such as safety (e.g., avoiding obstacles), reactivity (e.g., responding to events), and progress (e.g., patrolling). For robot missions specified in temporal logic, we can apply controller synthesis [16], [5] to either compute controllers that fulfill the given missions or show the non-existence of such discrete controllers.

Recently, the focus shifted towards optimizing the synthesized controller based on some given cost functions to improve the quality of continuous robot behaviors in addition to the correctness guarantee. A method by Smith et al. [17] automatically generates an optimal trajectory that achieves a mission specified in Linear Temporal Logic (LTL) [6], while minimizing the maximum time between reaching states that satisfy some optimization subformula. Wolff et al. [18] introduce a method for synthesizing an optimal controller from a temporal logic specification that minimizes the weighted average transition cost for a given transition system. Work by Karaman et al. [9], [10] aims at generating control laws that exhibit minimal control effort and mission time from tasks specified in LTL by integrating model checking with control optimization. In addition to temporal logic, quantitative techniques can also be used to provide "soft constraints" on the mission specifications. Bloem at al. [4] present a technique to analyze games with quantitative objectives. The reactive synthesis problem is typically reduced to solving a game between two players, so the introduction of quantitative objectives allows to naturally extend this concept to synthesize controllers of good quality.

All of these approaches however are not fully suitable for the optimization of continuous robot behavior in adversarial environments. They either consider non-reactive robot missions, or optimize towards a cost metric that is unsuitable for many robotics applications (e.g., the average cost per transition in a quantitative game).

In this paper, we solve the problem of synthesizing optimal robot controllers for an adversarial environment. The specification for the robot is expressed as an LTL formula. We base our approach on a new cost metric that captures desired robot behavior in such an environment. This metric is *two-dimensional*. The first dimension describes the number of phases in which the robot has to wait for the environment to perform progress towards satisfying the governing LTL formula. The second dimension is the cost of the actions performed by the robot to reach its goal. Both cost dimensions are considered *per reaching a goal*, rather than per discrete transition, as typically the case in quantitative games. We strive for minimizing both dimensions for an optimal controller, but these two aims are typically in conflict. As a remedy, we allow the user to choose an arbitrary preference relation over such cost tuples, and to declare what optimality means for the scenario under consideration.

Our starting point is the robot mission planning framework by Kress-Gazit et al. [15], in which the environment and robot behaviors are discretized and abstracted into sets of propositions with Boolean truth values (Definition 2.2-2.3). Whenever the existence of a correct controller is determined from the LTL specification, a discrete controller is synthesized. The discrete transitions of the controller

All authors are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, 14853, USA {gj56, rwe48, hadaskg}@cornell.edu. Rüdiger Ehlers is also affiliated with the Department of Electrical Engineering & Computer Sciences, University of California at Berkeley, CA, USA. This work was partially supported by NSF ExCAPE CCF-1139025 and NSF CAREER CNS-0953365.

are implemented with low-level continuous controllers to yield the desired robot behavior. Since in that framework, the continuous metric information is not considered when synthesizing the controller, the resulting behavior can be suboptimal in the continuous space. In previous work [8], we argued two reasons for the suboptimality: (i) The efficiency of the synthesis algorithm used in [5], [15] comes at the price of constructing controllers in which the robot's goals are followed in a fixed order, and (ii) the discrete controller is constructed by minimizing the number of discrete transitions needed to reach each robot goal, which need not minimize the cost in the continuous space. To address both points, we presented an algorithm that incorporates continuous metrics into controller synthesis [8]. Whenever environment properties change, a new controller is generated and then executed to yield desired continuous trajectories. In settings with a frequently changing environment, the robot might however get into livelock due to switching paths, and thus this solution is not well-suited for adversarial environments.

The combination of our new two-dimensional cost notion and the corresponding synthesis algorithm that we present in this paper solves all of these problems and in addition introduces the notion of *delay cost*. In adversarial environments, the strictest possible assumptions that we can make about the environment behavior are often only *liveness assumptions*. Whenever the robot has to rely on these being met, the length of a waiting time is not under the control of the robot. Assigning a cost to the action of waiting would thus lead to all possible behavior of the robot having the same worst-case cost (namely ∞), and thus rendering them effectively equal, which would prevent us from performing any kind of optimization of the robot's behavior. By distinguishing between waiting and transition cost, and only counting the number of *waiting phases* for the robot, we can efficiently optimize robot behavior in the continuous space in adversarial environments. On a technical level, we synthesize optimal controllers by a modification of the Generalized Reactivity(1) synthesis algorithm by Bloem et al. [5]. Despite the fact that we support non-integer costs for the actions of the robot, our new algorithm still has the possibility to use symbolic data structures such as binary decision diagrams (BDDs) for reasoning about the game, which is the source of the good scalability of Generalized Reactivity(1) synthesis.

The outline of this paper is as follows. In Section II, the preliminaries are defined. In Section III we introduce the optimal reactive controller synthesis problem. Section IV describes the algorithm for solving the problem. Section V demonstrates and discusses the performance of the algorithm.

II. Preliminaries

Definition 2.1 (Robot Workspace: P, τ): For the scope of this paper, we assume that a robot operates in some workspace that consists of a set of *regions* P. The possible transitions between the regions are given by an adjacency relation $\tau \subseteq P \times P$.

Definition 2.2 (Environment Input: X [8]): The status of the environment is abstracted with a set of binary environment propositions $\mathcal{X} = \{x_1, ..., x_n\}$ sensed by the robot's physical sensors. An environment input $X \subseteq \mathcal{X}$ is defined by the subset of environment propositions that are true. For example $\mathcal{X} = \{\text{DoorClosed, PathBlocked}\}$ is the set of environment propositions corresponding to a closed door and a blocked path in the environment that can be detected by a camera. $X = \{\text{DoorClosed}\}$ means that the robot senses a closed door, but not a blocked path. It is assumed that the robot's physical sensors provide the correct representation of the environment behavior, i.e. the low-level sensor processing routines eliminate any noise or non-determinacy in perception.

Definition 2.3 (System Output: Y [8]): The actions of the system are represented by the valuations of a set of binary system propositions $\mathcal{Y}$, which indicate the position, action, and status of the robot. Based on the partition of the workspace, the robot position is discretized using a set of binary region propositions $Reg = \{R_1, ..., R_m\}$. We define R_i to be true if and only if the robot is inside some region p_i. We define a mapping function $M(R_i) = p_i$ and its inverse $M'(p_i) = R_i$. Notice that exactly one R_i is true at any time. Robot actions, such as raising a flag, are also abstracted with a set of binary propositions $Act = \{a_1, a_2, ...\}$. a_i is true if the robot is performing the action and false otherwise. In addition, a set of auxiliary propositions Aux is used to represent all other properties of the robot such as whether the robot is carrying an item or not. We define the set of system propositions to be $\mathcal{Y} = Reg \uplus Act \uplus Aux$. A system output $Y \subseteq \mathcal{Y}$ is defined by the subset of system propositions that are true. For example $Y = \{R_3, Wave\}$ means that the robot is in region $M(R_3) = p_3$ and is waving.

Definition 2.4 (LTL Syntax and Semantics [5], [6], [8]): The syntax of Linear Temporal Logic (LTL) is defined over a set of atomic propositions $AP = \mathcal{X} \cup \mathcal{Y}$, a set of Boolean operators $\{\neg$ ("not"), $\wedge$ ("and")$\}$, and a set of temporal operators $\{\bigcirc$ ("next"), $\mathcal{U}$ ("until")$\}$. The syntax for the logic is then defined recursively as follows.

$$\varphi ::= true \mid z \in AP \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi$$

The additional Boolean operators $\vee$ ("or"), $\rightarrow$ ("implies") and $\leftrightarrow$ ("if and only if") are defined using $\neg$ and $\wedge$. We also define the temporal logic operators $\diamondsuit\ \varphi$ ("eventually φ") as $true\,\mathcal{U}\,\varphi$, and $\square\,\varphi$ ("always φ") by $\neg \diamondsuit \neg\varphi$.

A word $\sigma = \sigma_0\sigma_1\sigma_2 \ldots$ over the alphabet 2^{AP} is an infinite sequence of truth assignments to propositions of AP. We call σ a model of an LTL formula if it satisfies the LTL formula at position $i = 0$. The formula $\bigcirc \varphi$ is true at position i of σ if the formula φ is true at position $i + 1$. The formula $\diamondsuit \varphi$ is true at position i if φ holds in at least one position $\geq i$ for σ. In addition, $\square\varphi$ holds at position i if φ is true for all positions $\geq i$ in σ. All sub-formulas without a temporal operator are interpreted propositionally on σ_i.

Definition 2.5 (Mission Specification: φ): The specification of the robot is given as an LTL formula over propositions $AP = \mathcal{X} \cup \mathcal{Y}$:

$$\varphi = (\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \rightarrow_s (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s)$$

In this formula, φ_i^e, φ_i^s are conjunctions of initialization properties, φ_t^e, φ_t^s are the transition constraints, and φ_g^e and φ_g^s are conjunctions over properties of the form $\square\diamondsuit B_i$, where B_i is a Boolean formula over AP. The property B_i is called a robot goal if $\square\diamondsuit B_i$ is a conjunct in φ_g^s; if $\square\diamondsuit B_i$ is a conjunct in φ_g^e, then we call B_i an environment goal assumption. For the definitions of the other formulas, the reader is referred to Bloem et al. [5]. In the mission specification, the operator $\rightarrow_s$ is an *implication with strict semantics* [5].

Definition 2.6 (Finite State Automaton: A): As computation model for the robot controllers that we synthesize, we define a discrete controller to be a finite state automaton $A = (\mathcal{X}, \mathcal{Y}, Q, Q_0, \delta, L)$ where:

- $\mathcal{X}$ is the set of environment propositions
- $\mathcal{Y}$ is the set of robot propositions
- Q is the set of states
- $Q_0 \subseteq Q$ is the set of initial states
- $\delta : Q \times 2^{\mathcal{X}} \rightarrow Q$ is a transition function.
- $L : Q \rightarrow 2^{\mathcal{X}} \times 2^{\mathcal{Y}}$ is the labeling function of the states

We require that a finite state automaton is input-preserving, i.e., for every $q \in Q$ and $X \subseteq \mathcal{X}$, we have $L(\delta(q, X)) \cap \mathcal{X} = X$. Given a finite state automaton A, an initial state $q_0 \in Q_0$, and a sequence of environment inputs $\lambda_X = X_0, X_1, ...$, there exists a unique sequence $\pi = \pi_0\pi_1 \ldots \in Q^\omega$ with $\pi_0 = q_0$ and $\delta(\pi_j, \lambda_j) = \pi_{j+1}$ for all $j \in \mathbb{N}$. We call π the run of A for λ_X and q_0. The run induces a word $\sigma = \sigma_0\sigma_1 \ldots$ with $\sigma_i = L(\pi_i)$ for all $i \in \mathbb{N}$. A finite state automaton A satisfies a given LTL formula if all words that are induced by some combination of initial state and input sequence are models of the LTL formula. A mission specification over some atomic proposition set AP is said to be realizable if there exists a discrete controller that satisfies the given specification for the given partitioning of AP into $\mathcal{X}$ and $\mathcal{Y}$. We call a subset of states $S \subseteq Q$ a *strongly connected component (SCC)* if for every $q, q' \in S$, there exists a finite sequence of inputs $\lambda_X = X_0, X_1, ..., X_k$ such that $\delta(\delta(\ldots\delta(\delta(q, X_0), X_1), \ldots, X_{k-1}), X_k) = q'$. We call S maximal if no strict superset of S is an SCC in A.

Definition 2.7 (Game Structure [5]): In order to decide whether there exists a finite state automaton (Definition 2.6) that satisfies some given specification $\varphi = (\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \rightarrow_s (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s)$, the Generalized Reactivity(1) synthesis approach [5] reduces the problem to finding a winning strategy in a two-player game between a system (robot) and an environment.

Formally, we define a game structure (GS) to be a tuple $G = (AP, \mathcal{X}, \mathcal{Y}, \varphi_i^e \wedge \varphi_i^s, \rho_e, \rho_s, \varphi')$, where $AP = \mathcal{X} \cup \mathcal{Y}$ is the set of atomic propositions, $\mathcal{X}$ is the set of environment input propositions (as in Definition 2.2), $\mathcal{Y}$ is the set of system output propositions (as in Definition 2.3), $\varphi_i^e \wedge \varphi_i^s$ is the initial condition, and $\rho_e \subseteq 2^{AP} \times 2^{\mathcal{X}}$ and $\rho_s \subseteq 2^{AP} \times 2^{AP}$ are the transition relations of the environment and system over the positions 2^{AP} in the game, respectively. We call φ' the winning condition.

Bloem et al. [5] show how to build a game structure from a mission specification as described in Definition 2.5 with the winning condition $\varphi' = \varphi_g^e \rightarrow \varphi_g^s$. At each round (transition) in the game, the two players take turns, with the environment moving first and then the system, to set the value of their corresponding propositions based on their transition relations. This process results in the an (infinite) *play*. Plays can be winning or losing for the system. The aim of the system player is to ensure that the play is a model of the LTL formula φ'. The set of positions in the game from which the system player has a strategy to ensure that every possible play satisfies φ' is the winning set of positions. For realizable specifications, positions that satisfy $\varphi_i^e \wedge \varphi_i^s$ are winning for the system.

Definition 2.8 (μ-calculus [12], [5]): The winning set of positions of a two-player game can be captured with a formula in modal μ-calculus, which extends propositional modal logic with least and greatest fixpoint operators μ and ν [12]. Given a Boolean formula ψ over a set of propositions AP and a set $\Pi \subseteq AP$, we say that Π satisfies ψ, denoted by $\Pi \models \psi$, if setting each variable in Π to true and each variable in $AP \setminus \Pi$ to false leads to a valuation that satisfies ψ. The semantics of μ-calculus formulas over some game structure $G = (AP, \mathcal{X}, \mathcal{Y}, \varphi_i^e \wedge \varphi_i^s, \rho_e, \rho_s, \varphi')$ is defined recursively as described by Bloem et al. [5]:

- A Boolean formula ψ is interpreted as the set of positions $[\![\psi]\!]$ in which ψ is true: $[\![\psi]\!] = \{v \in 2^{AP} \mid v \models \psi\}$.
- The modal operator $\circledcirc$ is defined as: $[\![\circledcirc\psi]\!] = \{v \in 2^{AP} \mid \forall X \subseteq \mathcal{X}\ \exists Y \subseteq \mathcal{Y} : ((v, X) \in \rho_e) \rightarrow (((v, (X, Y)) \in \rho_s) \wedge ((X \cup Y) \in [\![\psi]\!]))\}$. In words, this is the set of positions v from which the system can force the play to reach a position in $[\![\psi]\!]$ without violating ρ_s, no matter which valid move the environment player makes from v.
- Let $\Psi(U)$ denote a μ-calculus formula Ψ with free variable U. We set $[\![\mu U.\Psi(U)]\!] = \cup_i U_i$ where $U_0 = \emptyset$ and $U_{i+1} = [\![\Psi(U_i)]\!]$. For monotone functions Ψ, this is a least fixpoint operation, computing the smallest set of positions U satisfying $U = \Psi(U)$.
- $[\![\nu U.\Psi(U)]\!] = \cap_i U_i$ where $U_0 = 2^{AP}$ and $U_{i+1} = [\![\Psi(U_i)]\!]$. For monotone functions, this is a greatest fixpoint operation, computing the largest set of positions U satisfying $U = \Psi(U)$.

By the results of Bloem et al. [5], for the winning condition $\varphi' = \varphi_g^e \rightarrow \varphi_g^s = (\square\diamondsuit B_1^e \wedge \ldots \square\diamondsuit B_m^e) \rightarrow (\square\diamondsuit B_1^s \wedge \ldots \square\diamondsuit B_n^s)$, the set of winning positions for the system in a game structure can be characterized by the μ-calculus formula $\varphi_{win} = \nu W.\bigwedge_{j=1}^{n} \mu V.\bigvee_{i=1}^{m} \nu U.N_{ij}$, where $N_{ij} = (B_j^s \wedge \circledcirc W \vee \circledcirc V \vee \neg B_i^e \wedge \circledcirc U)$.

For $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$, the greatest fixpoint $\nu U.N_{ij}$ characterizes the set of positions from which the system player can force the game to either (1) stay indefinitely in positions satisfying $\neg B_i^e$, thus falsifying the left-hand side of the implication $\varphi_g^e \rightarrow \varphi_g^s$, or (2) to reach a position in the set $Q_{win} = [\![B_j^s \wedge \circledcirc W \vee \circledcirc V]\!]$. The two outer fixpoints extend this idea to arbitrarily long sequences of steps and the requirement that along the way,

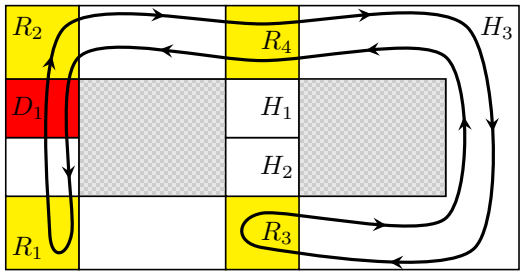


Fig. 1: The workspace of the motivating Example 3.1 and a graphical representation of the non-optimal robot behavior of a robot controller that the synthesis framework in [15] would compute.

we must stay within the set of positions that we did not yet find to be losing. The disjunction over the environment goal assumptions and the conjunction over the system goals makes sure that they are all taken into account. If the initial positions are winning, it is possible to extract an automaton that realizes the specification from the prefixpoints of this formula. Details of this process are given by Bloem et al. [5].

III. Problem Formulation

In this section we formalize our cost metrics for which we solve the optimal reactive controller synthesis problem in the next section. Our cost definition is *two-dimensional*. The **delay cost** due to potentially adversarial environment behavior captures the situation in which the robot is temporarily prevented from achieving its goals. The **transition cost**, as in earlier work [8], is used to represent the distance traveled by robot, energy consumption, or some other cost that accumulates over time.

Example 3.1 (Motivating Example): Figure 1 shows a map that defines a robot workspace. The regions of interest are labeled by R_1, R_2, R_3, R_4, D_1, H_1, H_2, and H_3. The two regions with checkers are obstacles. An environment proposition "*close*" captures the behavior of a door represented by the region D_1. The robot is asked to visit the regions R_1, R_2, R_3, and R_4 infinitely often, and to not go through region D_1 if *close* is true. The goal assumption on the environment is that it always eventually sets *close* to false so that the robot can go through region D_1. The following LTL formula defines part of the safety and goal requirements for the mission specification. The safety requirement that constrains the robot to only perform transitions between regions of the workspace that are adjacent is omitted.

$$\varphi = \square\diamondsuit(\neg close) \rightarrow (\square(\neg(D_1 \wedge close)) \wedge \square\diamondsuit R_1 \wedge \square\diamondsuit R_2 \wedge \square\diamondsuit R_3 \wedge \square\diamondsuit R_4) \quad (1)$$

Definition 3.1 (Modified Mission Specification: $\hat{\varphi}$): For some formula $\varphi_g^s = \square\diamondsuit B_1 \wedge \square\diamondsuit B_2 \wedge ... \square\diamondsuit B_n$ that represents the robot goals in a specification, a controller synthesized from the specification using the Generalized Reactivity(1) synthesis approach [5] achieves its goals in the order of $B_1, B_2, ..., B_n, B_1, ...$ To avoid this inherent ordering, we translate a mission specification $\varphi = (\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \rightarrow_s (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s)$ with the set of robot goals $\Phi_g = \{B_1, B_2, ..., B_n\}$ to a modified mission specification $\hat{\varphi}$ with only one robot goal. For this, we add some propositions $P_{mem} = \{m_{B_1}, m_{B_2}, ..., m_{B_n}, m_{B_{all}}\}$ to the system's output propositions and define the modified mission specification as follows:

$$\hat{\varphi} = \left(\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e\right) \rightarrow \left(\varphi_i^s \wedge \varphi_t^s \wedge \varphi_m \wedge \left(\bigwedge_{B_i \in \Phi_g} \varphi_{B_i}\right) \wedge \square\diamondsuit m_{B_{all}}\right) \quad (2)$$

In this formula, we have:

$$\varphi_m = \square\left(\left(\bigwedge_{m_{B_i} \in P_{mem}} m_{B_i}\right) \leftrightarrow m_{B_{all}}\right)$$

$$\forall B_i \in \Phi_g, \varphi_{B_i} = \square(B_i \wedge \neg m_{B_{all}} \rightarrow \bigcirc m_{B_i}) \wedge \square(m_{B_{all}} \rightarrow \bigcirc \neg m_{B_i}) \wedge \square(m_{B_i} \wedge \neg m_{B_{all}} \rightarrow \bigcirc m_{B_i}) \wedge \square(\neg m_{B_i} \wedge \neg B_i \rightarrow \bigcirc \neg m_{B_i})$$

The modified specification allows us to express multiple robot goal requirements with only one goal, at the expense of more propositions and a larger state space. As specified by φ_m, all memory propositions in P_{mem} need to be true at the same time for $m_{B_{all}}$ to be true, indicating that all robot goals have been satisfied once. However, the order in which the robot goals are satisfied is not specified. Once all of the goals have been reached, φ_{B_i} sets the propositions $m_{B_1}, m_{B_2}, ..., m_{B_n}$ to false. All robot goals have to be satisfied infinitely often in order to satisfy $\square\diamondsuit m_{B_{all}}$.

A. Delay Cost Due to Adversarial Environment Behavior

For some mission specifications, the environment can choose a behavior such that the robot is temporarily prevented from achieving its goals. In Example 3.1, when the robot decides to go from R_1 to R_2, one possibility for doing so is to take the left-most path, while waiting in front of the door if *close* is true. Since it is an environment goal to always eventually set *close* to false, and the robot only needs to work correctly in environments that satisfy their goals, the robot can assume to be able to travel from R_1 to R_2 through this path. However, the time it takes for the robot to wait for the door to open could be arbitrarily long.

In scenarios in which waiting is to be minimized, such a path may not be the best path for the robot. This is captured by the delay cost of a finite state automaton that we give in Def. 3.2. We use the simplified finite state automaton shown in Fig. 2 for illustration. Each state in the automaton is labeled by a state name as well as some associated region proposition that is true in that state. The transitions are labeled with constraints over the environment proposition *blocked*. The grey state represents an *m-state*. We call a state in which the proposition $m_{B_{all}}$ is true an *m-state*. Other labels are omitted for clarity. In this example, there are two doors that can temporarily prevent the robot from proceeding.

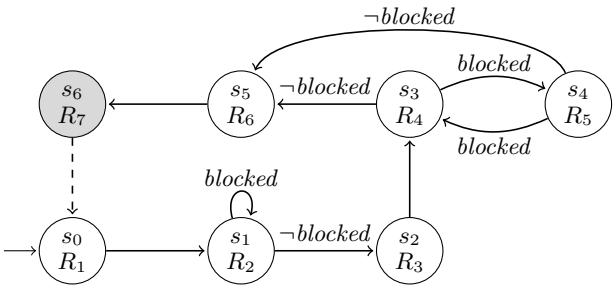


Fig. 2: A simplified finite state automaton

Definition 3.2 (Delay Cost: J_D): Let $\pi = \pi_0, \pi_1, \ldots, \pi_k$ or $\pi = \pi_0, \pi_1, \ldots$ be a finite or infinite sequence of states. We define a delay cost that captures the number of phases in which the robot may have to wait for the environment to fulfill its goal assumptions along π. Given a finite state automaton A satisfying a modified mission specification, we find the delay cost $J_D(\pi)$ by the following steps. We first remove the outgoing transitions from all *m-states*, shown as the dashed transitions in Fig. 2. Then we find the set $Q_D^A = \{Q_{D_1}, Q_{D_2}, ..., Q_{D_k}\}$ of maximal strongly connected components in A. Intuitively, every element Q_{D_i} captures a set of states in which the automaton can wait for some external event an indefinite number of steps. For the automaton shown in Fig. 2, we have $Q_D^A = \{\{s1\}, \{s3, s4\}\}$. The delay cost of π is then the number of elements in Q_D^A through which π leads. For the automaton in Fig. 2, for a sequence of states π from s_0 or s_1 to s_6, we have $J_D(\pi) = 2$, for π from s_2, s_3, or s_4 to s_6, we have $J_D(\pi) = 1$, and for π from s_5 or s_6 to s_6, we have $J_D(\pi) = 0$.

B. Transition Costs in the Continuous Space

In order to obtain robot behavior that is efficient when implemented continuously, we need to take the cost of performing actions (such as motion) by the robot into account. The default optimization criterion of the Generalized Reactivity(1) synthesis algorithm [5] is to minimize the number of discrete steps towards the goal, where waiting periods as defined above are counted as a single step. In the example workspace in Figure 1, in order to go from R_2 to R_3, since the path through H_3 has fewer discrete transitions (four transitions) than the path through H_1 and H_2 (five transitions), the former path is chosen even though it is longer. To counter this problem and in addition allow assigning a cost to non-motion actions by the robot (such as raising a flag), we introduce a transition cost notion below. Our synthesis algorithm can take it into account and thus optimize towards low-cost executions of the robot controller.

Definition 3.3 (Action Cost Map: w): Given a set of atomic propositions AP, we call a function $w : 2^{AP} \times 2^{AP} \to \mathbb{R}_{\geq 0}$ an action cost map.

Thus, an action cost map assigns to transitions between valuations of the variables AP a cost value. As AP includes the environment input and also Reg, this definition is very flexible: it allows to assign cost values to region changes by the robot and to make the cost of an action dependent on the input.

After defining the action cost for a single step in the execution of a robot controller, we turn towards summing these up for the execution of the controller. This leads to the transition cost notion that we optimize for in our synthesis algorithm.

Definition 3.4 (Transition Cost: J_T): Let $\pi = \pi_0, \pi_1, \ldots, \pi_k$ or $\pi = \pi_0, \pi_1, \ldots$, be a finite or infinite sequence of states. We define the accumulated transition cost of π to be:

$$c_t(\pi) = \sum_{i=0}^{b} w(\pi_i, \pi_{i+1}), \tag{3}$$

where $b = k - 1$ if π is finite and $b = \infty$ otherwise.

C. Cost Preference Relation

As our cost metric in this paper is two-dimensional, in order to synthesize *one* optimal controller, it needs to be defined which cost tuple values for executions of the robot controller to be synthesized are preferred over others. Our delay costs are from the domain $\mathbb{N}$, the transition cost is from the domain $\mathbb{R}$ extended by ∞ to allow denoting an unbounded transition cost. We say that some comparator $\leq_c$ is *sane* if and only if:

1) it respects the standard orders $\leq$ over $\mathbb{N}$ and $(\mathbb{R} \cup \{\infty\})$, i.e., for all (a, b) and $(a', b') \in \mathbb{N} \times (\mathbb{R} \cup \{\infty\})$, we have that $(a, b) \leq_c (a', b')$ if $a \leq a'$ and $b \leq b'$, and
2) it respects the standard order $\leq$ for the transition cost in the limit, i.e., if we have some cost tuple (a, b) and an infinite set of cost tuples $\mathcal{C} = \{(a', b_1), (a', b_2), (a', b_3), \ldots\}$ such that for all $i \in \mathbb{N}$, we have $(a, b) \leq_c (a', b_i)$ and there is no upper bound on the transition cost in $\mathcal{C}$, then we also have $(a, b) \leq (a', \infty)$.

For the scope of this paper, we assume that some sane cost preference relation is defined by the user, and we will define a concrete relation in Section V for our example.

D. Optimal Reactive Controller Synthesis Problem

Given some finite-state state automaton $A = (\mathcal{X}, \mathcal{Y}, Q, Q_0, \delta, L)$ that satisfies some modified specification $\hat{\varphi}$, we define a cost annotation $g : Q \to \mathbb{N} \times (\mathbb{R} \cup \{\infty\})$ to map every state q to a cost tuple $(c_d(q), c_t(q))$ such that no path π starting from q and ending at the first m-state after starting in q (or being infinite if no m-state is visited along π after starting in q) has some cost tuple $(c_d(\pi), c_t(\pi))$ such that $(c_d(\pi), c_t(\pi)) \not\leq_c (c_d(q), c_t(q))$. Intuitively, a cost annotation declares the worst case cost that may be observed along a path from a state q to the next m-state.

To simplify the controller synthesis process in the next section, we use a slightly modified version of the cost mapping for our optimal controller synthesis problem definition below. In particular, we let the robot change the environment goal that it is currently waiting for free of charge, i.e., we do not count switching between SCCs if this only changes what the robot is waiting for without making progress towards the

goal, provided that this change does not induce some non-zero transition cost. As it is not observable at runtime which environment goal the robot is waiting for, this modification is justified from a practical point of view.

We call a finite-state automaton A and a cost annotation g optimal for $\hat{\varphi}$ if for all $q \in A$, there exists no other finite-state automaton $A' = (\mathcal{X}, \mathcal{Y}, Q', Q'_0, \delta', L')$ that also satisfies $\hat{\varphi}$ and a corresponding cost annotation g' such that for some state $q' \in Q'$, we have $L(q) = L'(q')$, but $g(q) \not\leq_c g'(q')$.

We define the optimal controller synthesis problem as follows. Given a modified specification $\hat{\varphi}$ and an action cost map w, we search for an automaton that satisfies $\hat{\varphi}$ and is optimal for some cost annotation, or get the result that no such automaton exists.

Note that with this optimization criterion, we ask for a controller that greedily optimizes towards the next m-state (i.e., towards reaching all of the goals). The intuition for not optimizing for the time after that is that we do not know when the robot will go out of service – asking the robot to reach its goals infinitely often, as we do when applying Generalized Reactivity(1) synthesis for robotics applications, after all serves as abstraction for this unknown time instant, and greedily optimizing towards reaching the next m-state is reasonable under this uncertainty.

IV. Synthesis algorithm

In this section, we outline how to incorporate our two-dimensional cost definition into a synthesis algorithm. We start from the classical Generalized Reactivity(1) synthesis algorithm [5] and extend it accordingly.

Let $G := (AP, \mathcal{X}, \mathcal{Y}, \varphi_i^e \wedge \varphi_i^s, \rho_e, \rho_s, \varphi')$ be a game structure with $\varphi' = (\square\diamondsuit\varphi_1^e \wedge \ldots \wedge \square\diamondsuit\varphi_m^e) \rightarrow (\square\diamondsuit\varphi_1^s \wedge \ldots \wedge \square\diamondsuit\varphi_n^s)$ being the winning condition for the system player in this game structure. By the results of [5], we can compute the set of winning positions for the system player in the game structure by the following formula, which has been simplified for the case of having joined all goals into one (see Definition 3.1):

$$\nu W.\mu V.\bigvee_{i=1}^{m} \nu U.(\varphi^s \wedge \textcircled{\scriptsize$\diamond$}(W) \vee \textcircled{\scriptsize$\diamond$}(V) \vee (\neg\varphi_i^e) \wedge \textcircled{\scriptsize$\diamond$}(U))$$

In the outermost fixpoint operator application, we shrink a set of positions in the game until only those remain from which the system player can enforce to win. The $\mu V.(\ldots)$ part of the formula computes, step-by-step, the set of positions from which the system player can ensure that it either eventually reaches the goal and can enforce that the successor position from there is winning (i.e., is in W), or eventually some liveness assumption on the environment is not satisfied. The analysis of the game is mainly performed by the $\textcircled{\scriptsize$\diamond$}$ operator, which takes a set of positions O and computes another set of positions O' from which the system can enforce visiting O after one move of each player without violating the system player's safety guarantees, as long as the environment satisfies its safety assumptions. As positions in V are found in increasing distance to the goal, we are guaranteed to obtain a winning strategy if we always let it move to positions that are found as early as possible in V after W has stabilized.

Adding cost information does not change the set of winning positions in the game, as it does not influence whether the system can satisfy its specification or not. However, it does change the strategy that we want to obtain, and thus the finite-state automaton to be computed from the strategy.

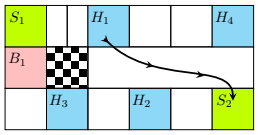
We incorporate our two-dimensional cost metric in the synthesis step by letting the fixpoint computation in the formula above work over pairs of positions and corresponding cost annotations. The $\textcircled{\scriptsize$\diamond$}$ operator then updates the transition cost to the goal along with the position from which we can get closer to the goal. The delay cost is incorporated by adding one to it whenever taking a transition that is found by the $(\neg\varphi_i^e) \wedge \textcircled{\scriptsize$\diamond$}(U)$ part of the formula (but not by the others). This case corresponds to waiting for the environment to satisfy φ_i^e. As transitions that can be used in such a waiting period are found at the same time, the delay cost is only added once per waiting period, instead of adding it for every transition. By preferring transitions to positions for which smaller cost annotations exist, we then ensure the optimality of the implementation that we are getting during the automaton building phase of the synthesis process.

V. Example and Result

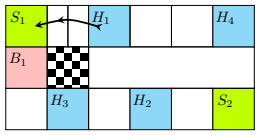
An example of a robot delivering milk in a neighborhood is used to demonstrate the performance of the synthesized controller. The map of the robot workspace is shown in Fig. 3. Moving from one region to another is considered to be a discrete transition. The area with checkers is an obstacle. The robot is equipped with sensors to detect whether the milk bucket is empty or not. The robot can also detect if the bridge (red region B_1) is blocked. The mission of the robot is to visit all four households (blue regions H) to deliver milk as long as the milk bucket is not empty. If it is, the robot must go to one of the stations (green regions S_1, S_2) to fill up the bucket and continue delivering. In addition, the robot cannot go through the bridge if it is blocked, which does not happen all the time. In this example, we prioritize minimizing delay cost over minimizing transition cost. Formally, $(c_d^1, c_t^1) \leq_c (c_d^2, c_t^2)$ if (i) $c_d^1 < c_d^2$ or, (ii) $c_d^1 = c_d^2$ and $c_t^1 \leq c_t^2$. We define the action cost map to assign cost to motion between regions by taking the metric centroid distance between the adjacent regions from/to which the robot moves.

We implemented our approach using binary decision diagrams (BDDs) as symbolic reasoning engine and integrated it into the Linear Temporal Logic Mission Planning (LTLMoP) toolkit [7]. We performed robot controller synthesis for the milk delivery example using both the algorithm introduced in this paper and the standard Generalized Reactivity(1) Synthesis algorithm [5] on a computer with an Intel i5@2.60GHz processor and 8GB of RAM. The computation time is 98.0 seconds for synthesizing an optimal robot controller using the algorithm from this paper, and 72.6 seconds for standard GR(1) synthesis.

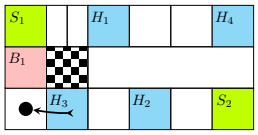
We demonstrate the behavior of the robot using a simulated environment. The milk bucket can be empty at any house region and is filled up instantaneously as long as it is in one of the station regions. Fig. 3a and 3b show the situation in which the robot is in H_1 and detects the empty bucket. Without the work in this paper, the robot in Fig. 3a goes to a further station with fewer discrete transitions. The optimal trajectory in Fig. 3b illustrates that the robot chooses to go to S_1 which is closer in continuous space. In Fig. 3c and 3d, the robot detects an empty bucket while in H_3. Fig. 3c shows the robot tries to drive to S_1 through B_1. However, the bridge (B_1) is blocked, so the robot has to wait for the bridge to open, which could take a long time. The trajectory in Fig. 3d demonstrates that the desired optimal behavior is to go to S_2 instead of S_1 when one prefers fewer delays at the expense of possible longer continuous trajectories. Fig. 3e shows the trajectories for the robot visiting all four house regions with a non-empty bucket and returning to the first visited house region. The trajectory is highly non-optimal due to the fixed goal order. In Fig. 3f, the robot visits all four house regions once (solid line) and then revisits all of them again in the reversed order (dashed line).



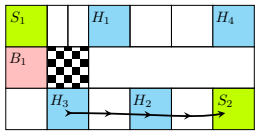
(a) The non-optimal trajectory from H_1 with an empty bucket. The robot went to S_2 because of fewer discrete transitions.



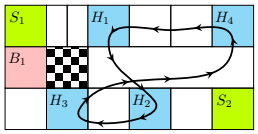
(b) The optimal trajectory from H_1 with an empty bucket. The robot went to S_1 since it is closer in continuous space.



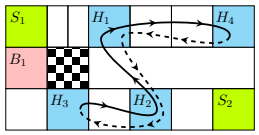
(c) The non-optimal trajectory from H_3 with an empty bucket and a blocked bridge. The robot chose to wait for the bridge to open.



(d) The optimal trajectory from H_3 with an empty bucket. The robot went to S_2 to avoid waiting.



(e) The non-optimal trajectory with a non-empty bucket. The robot visited all houses in a fixed non-optimal order.



(f) The optimal trajectory with a non-empty bucket. The robot visited the houses in the order of H_4, H_1, H_2, H_3 and then in the reversed order.

Fig. 3: The map and robot trajectories for the example in Section V. (The trajectories are slightly spread apart based on the actual simulation data for illustrative purposes.)

VI. Conclusion and Future Work

In this paper, we proposed a two-dimensional cost metric that consists of *delay cost* and *transition cost*. We allow the user to define a preference relation over tuples of these cost values and presented an algorithm to synthesize optimal controllers for this cost notion. The synthesized controller induces robot behavior that minimizes both the dependency on the environment behavior and the distance traveled, energy consumption, or whatever the user wishes to define the continuous transition cost over. Whenever these two aims are in conflict, the preference relation resolves the conflict, leading to a controller that is optimal with respect to the user preferences. Our experimental evaluation that we performed using a symbolic implementation of our algorithm shows the practical applicability of the approach.

References

[1] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics Automation Magazine*, 14(1):61–70, 2007.

[2] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Motion planning with hybrid dynamics and temporal goals. In *IEEE Conference on Decision and Control*, pages 1108–1115, Atlanta, GA, 2010. IEEE.

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

[4] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*, volume 5643 of *LNCS*, pages 140–156, 2009.

[5] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[6] E.A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier, 1995.

[7] C. Finucane, G. Jing, and H. Kress-Gazit. LTLMoP: Experimenting with language, temporal logic and robot control. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 1988 – 1993, 2010.

[8] Gangyuan Jing and Hadas Kress-Gazit. Improving the continuous execution of reactive LTL-based controllers. In *IEEE International Conference on Robotics and Automation*, May 2013.

[9] S. Karaman and E. Frazzoli. Linear temporal logic vehicle routing with applications to multi-UAV mission planning. *International Journal of Robust and Nonlinear Control*, 21(12):1372–1395, 2011.

[10] S. Karaman, R.G. Sanfelice, and E. Frazzoli. Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *47th IEEE Conference on Decision and Control*, 2008.

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

[12] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[13] H. Kress-Gazit, G. Fainekos, and G. Pappas. Where's Waldo? sensor-based temporal logic motion planning. In *IEEE International Conference on Robotics and Automation*, pages 3116–3121, 2007.

[14] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. From structured english to robot motion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2717–2722, 2007.

[15] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[16] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Berlin, 1992.

[17] S.L. Smith, J. Tumova, C. Belta, and D. Rus. Optimal path planning under temporal logic constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[18] E. Wolff, U. Topcu, and R. Murray. Optimal control with weighted average costs and temporal logic specifications. In *Robotics: Science and Systems*, Sydney, Australia, July 2012.

[19] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding horizon control for temporal logic specifications. *13th International Conference on Hybrid Systems: Computation and Control*, 2010.