Temporal Logic Robot Mission Planning for Slow and Fast Actions

Vasumathi Raman[1], Cameron Finucane and Hadas Kress-Gazit[2]

Abstract— **This paper addresses the challenge of creating correct-by-construction controllers for robots whose actions are of varying execution durations. Recently, Linear Temporal Logic synthesis has been used to construct robot controllers for performing high-level tasks. During continuous execution of these controllers by a physical robot, one or more low-level controllers are invoked simultaneously. If these low-level behaviors take different lengths of time to complete, the system will pass through several potentially unsafe intermediate states. This paper presents an algorithm that either generates a hybrid controller such that every continuous behavior of the robot is safe, or determines at synthesis time that the behavior may be unsafe. The proposed approach is implemented within the LTLMoP toolkit for reactive mission planning.**

I. Introduction

Robotics has seen an increase in the application of formal methods techniques for constructing controllers for high-level autonomous behaviors, including reactive conditions and infinite goals [1], [2], [3], [4]. Tasks involving such behaviors include search and rescue missions and the control of autonomous vehicles. The traditional approach to implementing high-level behaviors is to hard-code the high-level control and use path-planning and other low-level techniques during execution; with this method, it is not always known a priori whether an implementation fulfills the desired requirements. Formal methods provide guarantees that the implemented controller will produce the desired behavior.

Formal techniques that have been applied to high-level robot planning include model checking [5], [1], [2] and synthesis, in which a correct-by-construction controller is automatically synthesized from a formal task specification [3], [4]. Synthesis-based approaches take as input a discrete abstraction of the robot workspace and a temporal logic specification of the environment assumptions and desired robot behavior, and yield an automaton fulfilling the specification on the abstraction if the task is feasible. This automaton is viewed as a hybrid controller, calling low-level continuous controllers to actuate transitions between discrete states.

Example 1 *Consider an Aldebaran Nao robot, performing a task in the lab [6]. The available actions for this robot include motion of the arm (waving), a text-to-speech interface, and walking; walking between regions of interest takes significantly longer than any of the other actions.*

This work was supported by NSF CAREER CNS-0953365, ARO MURI (SUBTLE) W911NF-07-1-0216 and NSF ExCAPE

[1]V. Raman is with the Department of Computer Science, Cornell University, Ithaca, NY, USA vraman@cs.cornell.edu

[2]C. Finucane and H. Kress-Gazit are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, USA {cpf37,hadaskg}@cornell.edu

In the discrete abstraction of the above problem, the robot's state encodes the robot's current location and whether it is waving. Suppose the implementing automaton contains a discrete transition from the state where the robot location is region r_1 and it is not waving, to the state where the robot location is r_2 and it is waving. This discrete transition corresponds to executing two continuous controllers – one for motion and one for waving; the controller for waving takes less time to complete execution than the motion between rooms. In general, a single transition between states in the discrete problem abstraction requires executing low-level controllers with different completion times. The continuous implementation of the synthesized hybrid controller in [3] may result in a delayed response to events in the environment, and in unsafe continuous executions (Section III), even though the discrete automaton is provably correct.

This observation motivates a controller synthesis procedure that ensures safety of continuous execution for every discrete transition, while still allowing immediate reaction to the environment. This paper presents an algorithm that either generates a reactive hybrid controller such that every continuous behavior of the robot is safe, or determines at synthesis time that the behavior may be unsafe; the algorithm handles a class of specifications corresponding to a fragment of Linear Temporal Logic. This is the first work to address the challenges of continuous execution using low-level controllers of different execution times. The described approach is implemented within the Linear Temporal Logic MissiOn Planning (LTLMoP) toolkit [7].

II. Preliminaries

Applying formal methods tools to the robot control problem requires (a) a discrete abstraction of the problem in which the continuous reactive behavior of a robot is described in terms of a finite set of states, and (b) a temporal logic formalism for the specification, which in this work is Linear Temporal Logic (LTL) [5]. This section provides details on these components, as well as the synthesis algorithm.

A. Linear Temporal Logic

Syntax: Let AP be a set of atomic propositions. LTL formulae are recursively constructed from $\pi \in AP$ according to the grammar: $\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \Box\varphi \mid \Diamond\varphi$, where $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is "next", $\Box$ is "always", and $\Diamond$ is "eventually". The LTL operator $\mathcal{U}$ ("until") is omitted for this work. Boolean constants `True` and `False` are defined as $\texttt{True} = \pi \vee \neg\pi$ and $\texttt{False} = \neg\,\texttt{True}$. Conjunction ($\wedge$), implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$) are derived.

Semantics: The truth of an LTL formula is evaluated over

executions of a finite state machine representing the system. A state is an assignment of truth values to all propositions $\pi \in AP$. An execution σ is viewed as an infinite sequence of these states, and $\sigma \models \varphi$ denotes that σ satisfies formula φ:

- $\sigma \models \bigcirc \varphi$ iff φ is true in the second position in σ.
- $\sigma \models \Box \varphi$ iff φ is true at every position in σ.
- $\sigma \models \Diamond \varphi$ iff φ is true at some position in σ.
- $\sigma \models \Box \Diamond \varphi$ iff φ is true infinitely often in σ.

A finite state machine A satisfies φ if for every execution σ of A, $\sigma \models \varphi$. For a formal definition of the semantics, the reader is referred to [5]. LTL is well-suited to specifying *safety* ("something bad never happens") and *liveness* ("something good eventually happens") properties of high-level behavior: these correspond to LTL formulas with operators $\Box$ and $\Diamond$ respectively (the fragment of LTL used in this work supports liveness conditions of the form $\Box \Diamond$ instead).

B. Discrete Abstraction and Task Specification

The discrete abstraction of the high-level robot task consists of a set of propositions $\mathcal{X}$ whose truth value is determined by the robot's sensors and controlled by the environment (sensors and environment are used interchangeably in this paper), and a set of action and location propositions $\mathcal{Y}$ controlled by the robot (also referred to as "the system"). $\mathcal{X}$ and $\mathcal{Y}$ are sometimes called the input and output propositions respectively. $\mathcal{L} \subseteq \mathcal{Y}$ denotes the set of location propositions. The value of each $\pi \in \mathcal{X} \cup \mathcal{Y}$ is the abstracted binary state of a low-level black box component. For example, sensor propositions correspond to thresholded sensor values, and robot location propositions correspond to the robot's location with respect to a partitioning of the workspace.

Example 2 *Consider the workspace depicted in Fig. 4. The robot has one sensor, which senses a person (the corresponding proposition is π_{person}), and one action, which is to turn a camera on or off (π_{camera}). In addition, there are two possible locations, r_1 and r_2 (π_{r_1}, π_{r_2}), which are adjacent. The robot starts in room r_1 with the camera off. When it senses a person, it must turn on the camera. Additionally, the robot is required to visit room r_2 infinitely often.*

Here $\mathcal{X} = \{\pi_{person}\}$ and $\mathcal{Y} = \{\pi_{camera}, \pi_{r_1}, \pi_{r_2}\}$. Since the robot can be in exactly one of r_1, r_2 at any given time, the formulae $\varphi_{r_1} = \pi_{r_1} \wedge \neg \pi_{r_2}$ and $\varphi_{r_2} = \pi_{r_2} \wedge \neg \pi_{r_1}$ are used to represent the robot being in r_1 and r_2 respectively. The task in Example 2 corresponds to the following LTL specification:

$$
\begin{aligned}
& \varphi_{r_1} \wedge \neg \pi_{camera} \\
& \qquad \text{(Robot starts in region r1 with the camera off)} \\
\wedge \quad & \Box(\bigcirc \pi_{person} \Leftrightarrow \bigcirc \pi_{camera}) \\
& \qquad \text{(Activate the camera if and only if you see a person)} \\
\wedge \quad & \Box \Diamond (\varphi_{r_2}) \\
& \qquad \text{(Go to r2 infinitely often)}
\end{aligned}
$$

The possible motion of the robot in the workspace is determined by adjacency of the regions and the existence of controllers that will drive the robot from one region to another. This information is automatically encoded in the logic specification as a formula over location propositions, defining legal transitions between adjacent regions.

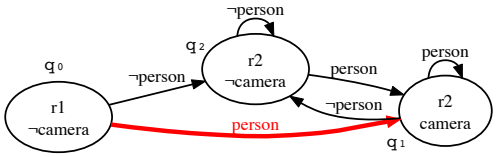


Fig. 1: Synthesized automaton for Example 2.

C. Synthesis

Given a task specification and a description of the workspace topology, this work considers formulae of the form $\varphi_e \Rightarrow \varphi_s$, where φ_e encodes assumptions about the environment's behavior and φ_s represents the desired robot behavior. φ_e and φ_s each have the structure $\varphi_e = \varphi_e^i \wedge \varphi_e^t \wedge \varphi_e^g$, $\varphi_s = \varphi_s^i \wedge \varphi_s^t \wedge \varphi_s^g$. The initial conditions φ_e^i and φ_s^i are non-temporal Boolean formulae over $\mathcal{X}$ and $\mathcal{Y}$ respectively, transition relations φ_e^t and φ_s^t represent safety assumptions on the environment and restrictions on the system, and φ_e^g and φ_s^g represent liveness assumptions on the environment and desired robot liveness behaviors, respectively.

An LTL formula φ is *realizable* if, for every time step, given a truth assignment to the environment propositions for the next time step, there is an assignment of truth values to the robot propositions such that the resulting infinite sequence of truth assignments satisfies φ. The synthesis problem is to find an automaton that encodes these assignments, i.e. whose executions satisfy φ.

Definition 1 *An* automaton *is a tuple $A = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma_\mathcal{X}, \gamma_\mathcal{Y})$ where*

- *Q is a set of states.*
- *$Q_0 \subseteq Q$ is a set of initial states.*
- *$\mathcal{X}$ is a set of inputs (sensor propositions).*
- *$\mathcal{Y}$ is a set of outputs (location and action propositions).*
- *$\delta : Q \times 2^\mathcal{X} \to Q \cup \{\perp\}$ is the deterministic transition relation. Informally, if $\delta(q,x) = \perp$, then the set of environment inputs x is disallowed in state q.*
- *$\gamma_\mathcal{X} : Q \to 2^\mathcal{X}$ is a transition labeling, which associates with each state the set of environment propositions that are true over incoming transitions for that state (note that this set is the same for all transitions into a given state). Note that if $q' \in \delta(q,x)$ then $\gamma_\mathcal{X}(q') = x$.*
- *$\gamma_\mathcal{Y} : Q \to 2^\mathcal{Y}$ is a state labeling, associating with each state the set of system propositions true in that state.*

Define $\gamma(q) = \gamma_\mathcal{X}(q) \cup \gamma_\mathcal{Y}(q)$ for $q \in Q$. Given a sequence of states $\sigma = q_0 q_1 q_2 \ldots$ where $q_0 \in Q_0$, define $\Gamma(\sigma) = \gamma(q_0)\gamma(q_1)\gamma(q_2)\ldots$. Let $\delta_\mathcal{X}(q) = \{x \in 2^\mathcal{X} \mid \delta(q,x) \neq \perp\}$, $\delta_\mathcal{Y}(q,x) = \gamma_\mathcal{Y}(\delta(q,x))$, and $\delta(q) = \{\delta(q,x) \mid x \in \delta_\mathcal{X}(q)\}$.

Definition 2 *Given $\varphi = (\varphi_e \Rightarrow \varphi_s)$, automaton $A_\varphi = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma_\mathcal{X}, \gamma_\mathcal{Y})$* realizes *$\varphi$ if $\forall \sigma = q_0 q_1 q_2 \ldots \in Q^\omega$ such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i)$, $\Gamma(\sigma) \models \varphi$.*

When restricted to LTL formulas with the form described above, the algorithm introduced in [8] permits synthesis in time polynomial in the size of the state space. Fig. 1 depicts the automaton synthesized for the above specification.

D. Continuous Execution

Given an automaton that implements a specification in a discrete abstraction of the problem, it remains to create a controller that implements the corresponding continuous behavior. For this, the automaton is viewed as a hybrid controller, wherein a transition between states is achieved by the activation of one or more atomic continuous controllers corresponding to each robot proposition. Consider Example 2, and the synthesized automaton in Fig 1. Consider the highlighted transition between states q_0, in which the robot is in r_1 with the camera off and no person sensed, and q_1, in which the robot has sensed a person and is in r_2 with the camera on. To execute this transition when a person is sensed in q_0, the hybrid controller described in [3] first activates a low-level motion controller to drive the robot from r_1 to r_2 (changing the values of π_{r_1} and π_{r_2}), and then calls another low-level controller to turn on the camera (changing π_{camera}) once the robot has crossed the boundary between r_1 and r_2.

The atomic controllers used satisfy the bisimulation property [9], which ensures that if the discrete robot model satisfies an LTL formula, then the continuous model also satisfies the same formula (e.g., the motion controllers are guaranteed to drive the robot from one region to another regardless of the initial state within the region). The feedback controllers presented in [10] and [11] are among several that satisfy this property. The reader is referred to [3] for a complete discussion of the hybrid controller, and to [7] for details on the incorporation of atomic controllers in LTLMoP.

III. Problem Formulation

In the discrete automaton generated by the above construction, several system propositions may change value over a single discrete transition, requiring the invocation of low-level controllers that take different amounts of time to execute. For clarity of presentation, assume that there are two kinds of low-level controllers – *fast* and *slow* – taking times t_F and t_S respectively, with $t_F < t_S$. More specifically, assume that motion is the only slow controller. The set of system propositions is partitioned based on the speed of the corresponding low-level controllers, into $\mathcal{Y}_S = \mathcal{L}$ and $\mathcal{Y}_F = \mathcal{Y} \backslash \mathcal{L}$ (i.e. location and non-location propositions). In Example 2, $\mathcal{Y}_F = \{\pi_{camera}\}$ and $\mathcal{Y}_S = \{\pi_{r_1}, \pi_{r_2}\}$.

Given a discrete transition between states (q, q'), if there exist location propositions $\pi_r \in \gamma_{\mathcal{L}}(q)$ and $\pi_{r'} \in \gamma_{\mathcal{L}}(q')$ where $r \neq r'$, then the motion controller for driving the robot from r to r' is activated first. The remaining controllers for all the propositions that change value over the transition, i.e. for $\pi_y \in [\gamma_{\mathcal{Y}}(q) \backslash \gamma_{\mathcal{Y}}(q') \cup \gamma_{\mathcal{Y}}(q') \backslash \gamma_{\mathcal{Y}}(q)] \backslash \mathcal{L}$, are only activated (or deactivated) once the robot had crossed the boundary between r and r'. Assuming instantaneous activation of the π_y controllers, this also completes the discrete transition (q, q'). Fig. 2(a) depicts the change in state for the transition depicted in Fig. 1, and how it corresponds to the progress of the continuous controllers.

This approach to continuous execution has two drawbacks when some controllers are non-instantaneous:

Delayed Response: The above approach of executing the slow action first (in this case motion) sometimes sacrifices responsiveness in exchange for safety of continuous execution under the assumption of instantaneous fast actions. It is usually desirable to respond to sensor inputs in a "greedy" manner, i.e. as fast as possible. For example, the camera should be turned on as soon as a person is sensed, regardless of the other actions to be performed. However, with this approach to continuous execution, the robot will not turn on its camera as soon as it senses a person, instead waiting until the transition to r_2 has been completed.

Unsafe Intermediate States: While continuous execution using the original approach is safe for instantaneous fast actions, it admits potentially unsafe intermediate states when fast actions are non-instantaneous. For example, if the low-level controller for turning on the camera is non-instantaneous, the continuous execution of the controller in Example 2 will pass through the intermediate state $q_0^{\bar{f}s}$ (not present in the discrete automaton) with $\gamma_{\mathcal{Y}}(q_0^{\bar{f}s}) = \{\pi_{r_2}\}$, as depicted in Fig. 2(b). Although in this example, $q_0^{\bar{f}s}$ is a safe state, this is not true in general. Furthermore, if the camera is activated simultaneously with the motion to allow immediate reaction to the sensor, the execution would pass through $q_0^{f\bar{s}}$ (Fig. 2(c)), which could be an unsafe state.

Example 3 *Consider Example 2 with the added safety:*
$\square(\neg(\pi_{camera} \wedge \pi_{r_1}))$ *(Do not activate the camera in r1)*

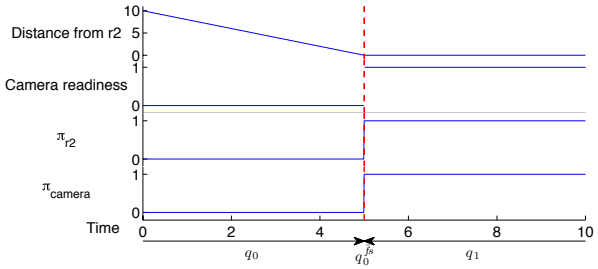
This requires that the camera never be turned on when in r_1. In this case, the intermediate state $q_0^{f\bar{s}}$ resulting from simultaneous activation of the controllers will be unsafe. In other words, the execution depicted in Fig. 2(b) is still safe while that in Fig. 2(c) is not. It is desirable to obtain a controller that guarantees safety of intermediate states that are not explicitly present in the synthesized automaton or checked during the existing synthesis process, but rather occur as artifacts of the continuous execution.

It may seem reasonable to circumvent these problems by requiring at most one robot action per transition; however, this could result not just in unnecessarily large automata, but also in newly unsynthesizable specifications. For instance, the specification in Example 2 would be unsynthesizable because the robot can never move from r_1 to r_2 if the environment alternates between person and no person (since the robot will have to toggle the camera on and off, and cannot move while doing so).

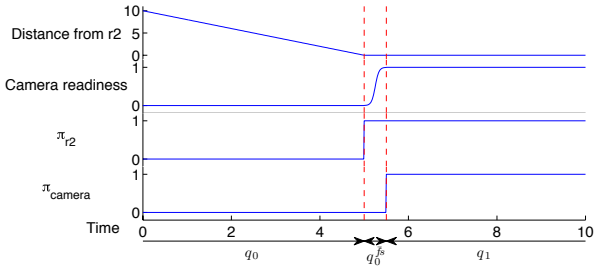
Given a discrete transition $(q, q'), \mathcal{Y} = \mathcal{Y}_F \cup \mathcal{Y}_S$, define $\gamma_F(q) = \gamma_{\mathcal{Y}}(q) \cap \mathcal{Y}_F$ and $\gamma_S(q) = \gamma_{\mathcal{Y}}(q) \cap \mathcal{Y}_S$. Let $q^{f\bar{s}}$ define the discrete state with $\gamma_{\mathcal{Y}}(q^{f\bar{s}}) = \gamma_{\mathcal{Y}_S}(q) \cup \gamma_{\mathcal{Y}_F}(q')$. This is the intermediate state in the transition between q and q', such that the fast actions have finished executing but the slow actions have not. Define

$$h(q, q') = \begin{cases} qq^{f\bar{s}} & \text{if } \gamma_{\mathcal{Y}_S}(q) \neq \gamma_{\mathcal{Y}_S}(q') \wedge \gamma_{\mathcal{Y}_F}(q) \neq \gamma_{\mathcal{Y}_F}(q') \\ q & \text{otherwise} \end{cases}$$

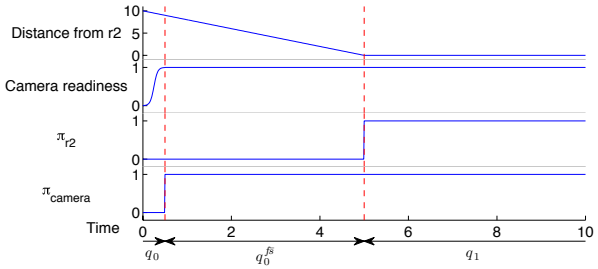
This is a function that returns the intermediate state $q^{f\bar{s}}$ if both slow and fast actions change over the transition (q, q').



(a) Motion completes first, instantaneous camera. This corresponds to the approach in [3] (assuming instantaneous fast actions).



(b) Actual execution of 2(a). Motion completes first, camera is non-instantaneous.



(c) Camera completes first in the new approach.

Fig. 2: Timing diagrams for continuous execution of the discrete transition in Fig. 1

Note that if only the slow actions or only the fast actions change, there is no intermediate state in the continuous execution. In Example 2, $h(q_0,q_1) = q_0 q_0^{f\bar{s}}$.

Definition 3 *Given* $A = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma_\mathcal{X}, \gamma_\mathcal{Y})$, *let* $H_A^{FS} = \{h(q_0,q_1)...h(q_i,q_{i+1})... \mid q_0 q_1 ... \in Q^\omega, q_{i+1} \in \delta(q_i)\}$.

H_A^{FS} defines the projection onto the set of discrete states Q of all continuous executions of automaton A when there are controllers of two completion times, and they are executed simultaneously to implement each discrete transition.

Problem 1 *Given* $\varphi, \mathcal{Y} = \mathcal{Y}_F \cup \mathcal{Y}_S$ *and a set of safe states* Q_{safe}, *the goal is to construct* A_φ *such that* $\forall \sigma \in H_{A_\varphi}^{FS}, \sigma \in Q_{safe}^\omega$ *(if such an automaton exists).*

Intuitively, the goal is to generate an implementing automaton such that every continuous execution contains only safe states. The set of safe states Q_{safe} can be arbitrarily defined – in this paper, it is the set of all states not explicitly excluded by the specified safety properties φ_e^t and φ_s^t.

IV. Synthesis for Fast and Slow Actions

This section presents a synthesis algorithm that guarantees correctness of continuous executions when simultaneously executing low-level controllers of two different completion times for every discrete transition. The synthesis is based on the algorithm in [8], and reduces the realizability problem to finding a winning strategy in a game played between the system (robot) and the (adversarial) environment. The two players alternate "moves", which correspond to setting values for their respective propositions according to their transition relations: the environment moves first in each time step, and is followed by the system. An infinite alternation of player moves is winning for the system if it either satisfies the system transition relation and liveness requirements, or prevents the environment assumptions from being fulfilled. The set of states from which there exists a winning strategy for the system is called the winning set of states. If the specification is realizable, every initial state admitted by $\varphi_s^i \wedge \varphi_e^i$ is winning for the system. On the other hand, if the environment can make moves to prevent the system from responding in a manner that satisfies φ_s, no automaton is generated. When this happens in LTLMoP, information about the cause of the unsynthesizability of the specification is displayed to the user [12], [13].

Formally, the set of states that are winning for the system can be characterized using the modal μ-calculus, which extends propositional modal logic with least and greatest fixpoint operators μ, ν [14]. Given a set of propositions P, $P \models \varphi$ denotes that truth assignments setting $\pi \in P$ to `True` and $\pi \notin P$ to `False` satisfy the Boolean formula φ. The syntax of μ-calculus formulae over A_φ is defined recursively:

- A Boolean formula φ is interpreted as the set of states $[\![\varphi]\!]$ in which φ is true, i.e. $[\![\varphi]\!] = \{q \in Q \mid \gamma(q) \models \varphi\}$. The set of states $[\![\varphi]\!]$ is defined inductively on the structure of the μ-calculus formula.
- The logical operator $\diamondsuit$ is defined as in [8]: $[\![\diamondsuit \varphi]\!] = \{q \in Q \mid \forall x \in \delta_\mathcal{X}(q), \delta(q,x) \in [\![\varphi]\!]\}$. In words, this is the set of states q from which the system can force the play to reach a state in $[\![\varphi]\!]$, regardless of what move the environment makes from q (i.e. for any $x \in \delta_\mathcal{X}(q)$). In Example 2, $[\![\diamondsuit \pi_{r_2}]\!]$ is the set of all states in which the robot can move to region r_2, regardless of what the environment does, so $[\![\diamondsuit \pi_{r_2}]\!] = \{q_0, q_1, q_2\}$ in Fig. 1.
- Let $\psi(X)$ denote a μ-calculus formula ψ with free variable X. $[\![\mu X.\psi(X)]\!] = \cup_i X_i$ where $X_0 = \emptyset$ and $X_{i+1} = [\![\psi(X_i)]\!]$. This is a least fixpoint operation, computing the smallest set of states X satisfying $X = \psi(X)$.
- $[\![\nu X.\psi(X)]\!] = \cap_i X_i$ where $X_0 = Q$ and $X_{i+1} = [\![\psi(X_i)]\!]$. This is a greatest fixpoint operation, computing the largest set of states X satisfying $X = \psi(X)$.

In [8], the set of winning states for the system is characterized by the μ-calculus formula $\varphi_{win} =$

$$\nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_n \end{bmatrix} . \begin{bmatrix} \mu Y.\left(\bigvee_{i=1}^m \nu X.(J_s^1 \wedge \diamondsuit Z_2 \vee \diamondsuit Y \vee \neg J_e^i \wedge \diamondsuit X)\right) \\ \mu Y.\left(\bigvee_{i=1}^m \nu X.(J_s^2 \wedge \diamondsuit Z_3 \vee \diamondsuit Y \vee \neg J_e^i \wedge \diamondsuit X)\right) \\ \vdots \\ \mu Y.\left(\bigvee_{i=1}^m \nu X.(J_s^n \wedge \diamondsuit Z_1 \vee \diamondsuit Y \vee \neg J_e^i \wedge \diamondsuit X)\right) \end{bmatrix}$$

where J_e^i is the i^{th} environment liveness ($i \in \{1,...,m\}$), and J_s^j is the j^{th} system liveness ($j \in \{1,...,n\}$). For $i \in$

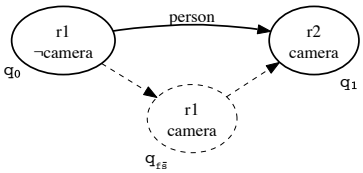


Fig. 3: Intermediate state with fast camera and slow motion

$\{1,...,m\}$ and $j \in \{1,...,n\}$, the greatest fixpoint $\nu X.(J_s^j \wedge \bigcirc Z_{j\oplus 1} \vee \bigcirc Y \vee \neg J_e^i \wedge \bigcirc X)$ characterizes the set of states from which the robot can force the game to stay infinitely in states satisfying $\neg J_e^i$, thus falsifying the left-hand side of the implication $\varphi_e \Rightarrow \varphi_s$, or in a finite number of steps reach a state in the set $Q_{win} = [\![J_s^j \wedge \bigcirc Z_{j\oplus 1} \vee \bigcirc Y]\!]$. The two outer fixpoints ensure that the robot wins from the set Q_{win}: μY ensures that the play reaches a $J_s^j \wedge \bigcirc Z_{j\oplus 1}$ state in a finite number of steps, and νZ ensures that the robot can loop through the livenesses in cyclic order. From the intermediate steps of the above computation, it is possible to extract an automaton that realizes the specification, provided every initial state is winning; details are available in [8].

To incorporate the relative execution times of the robot controllers, the synthesis algorithm is further constrained to generate only automata with safe intermediate states as follows. Given $\varphi, \mathcal{Y} = \mathcal{Y}_F \cup \mathcal{Y}_S$ and Q_{safe}, define the set:

$$FS_\varphi^1 = \{q \in Q \mid \forall x \in \delta_{\mathcal{X}}(q), \delta(q,x) \in [\![\varphi]\!], \gamma_{\mathcal{Y}_F}(q) \neq \gamma_{\mathcal{Y}_F}(q') \text{ and } \gamma_{\mathcal{Y}_S}(q) \neq \gamma_{\mathcal{Y}_S}(q'), \text{ and } q^{f\bar{s}} \in Q_{safe}\}$$

This is the set of states from which the system can in a single step force the play to reach a state in $[\![\varphi]\!]$ by executing actions of both fast and slow controller durations. In addition, the intermediate state $q^{f\bar{s}}$ in the corresponding continuous execution is safe. Also define

$$FS_\varphi^2 = \{q \in Q \mid \forall x \in \delta_{\mathcal{X}}(q), \delta(q,x) \in [\![\varphi]\!] \text{ and } (\gamma_{\mathcal{Y}_F}(q) = \gamma_{\mathcal{Y}_F}(q') \text{ or } \gamma_{\mathcal{Y}_S}(q) = \gamma_{\mathcal{Y}_S}(q'))\}$$

This is the set of states from which the system can force the play to reach a state in $[\![\varphi]\!]$ by executing actions of only one controller duration (fast or slow). In this case, there are no intermediate discrete states in the continuous execution.

Using the above two sets, define a new operator $[\![\bigcirc_{FS}\varphi]\!] = FS_\varphi^1 \cup FS_\varphi^2$. Informally, $\bigcirc_{FS}$ is the set of states q from which the system can force the play to reach a state in $[\![\varphi]\!]$, regardless of what move the environment makes from q, with the additional constraint that, if both fast and slow controllers are to be executed to implement a transition, the resulting intermediate state $q^{f\bar{s}}$ (depicted in Fig. 3) is safe.

Returning to Example 3, where the system safety condition includes "Always not camera in r_1" ($\square(\neg(\pi_{camera} \wedge \pi_{r_1}))$), a state in which the system senses a person is only in $[\![\bigcirc_{FS}\pi_{camera}]\!]$ if the robot can stay in the same region while turning on the camera. Recall that q_0 is the state in which the robot is in r_1 with the camera off. Observe that $q_0 \notin [\![\bigcirc_{FS}\pi_{camera}]\!]$ (recall that this means that in q_0, the robot cannot guarantee that the camera will be turned on in the next time step). This is because, if the environment sets π_{person} to true while the robot is still in r_1, the safety condition $\square(\neg(\pi_{camera} \wedge \pi_{r_1}))$ prevents the robot from turning on the camera before first moving to r_2, and so the camera cannot

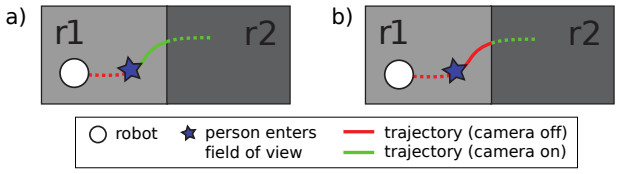


Fig. 4: Comparison of continuous trajectories and discrete events resulting from the two approaches for Example 2. a) Camera is turned on as soon as a person is sensed, according to the approach in this paper. b) When a person is sensed, motion is completed first, then camera turns on. This corresponds to the approach in [3].

be immediately activated since it would finish execution before the robot had left r_1. The corresponding specification is unrealizable under the new synthesis algorithm, whereas the original synthesis algorithm would return an automaton that included the transition (q_0, q_1) in Fig. 1. This difference is consistent with the observation that this transition is safe for the original execution in [3], under the assumption of instantaneous fast actions, but is unsafe if all action controllers are to be called simultaneously.

The proposed synthesis algorithm is accompanied by a new execution paradigm that calls all low-level controllers corresponding to a discrete transition simultaneously. Once a transition is begun, the controller ignores changes in the environment until the (slower) motion action completes.

V. Implications of New Approach

This section presents some implications of the new synthesis and execution approach in terms of changes in the observed behavior when compared with [3].

A. Immediate Response

Consider again the specification in Example 2, in which the robot has to move from its starting position r_1 to its destination r_2 and turn its camera on if it sees a person along the way. With the new execution paradigm, the hybrid controller turns the camera on immediately when a person is sensed. The trajectory that results from this controller is depicted in Fig. 4(a). (For the purpose of illustration, in this example we assume that the person, once visible to the robot, remains visible.) Using the previous approach to synthesis and execution, in which slow actions are executed before fast actions, this specification would result in undesired behavior: even if the robot sensed a person while in the middle of r_1, it would only react to it once it completed its movement to region r_2. This is depicted in Fig. 4(b). Furthermore, the person would still need to be sensed at the time of region transition, or else a different transition would be chosen and the person would effectively be ignored.

This problem, which arises in the previous approach, can be pre-empted by adding an extra safety requiring the system to momentarily stay in place when reacting to a person:

$$\square \; ((\pi_{person} \neq \bigcirc \pi_{person}) \Rightarrow ((\bigcirc \pi_{r_1} \Leftrightarrow \pi_{r_1}) \wedge (\bigcirc \pi_{r_2} \Leftrightarrow \pi_{r_2})))$$

(If the value of the person sensor changed, stay in place)

This requirement forces the synthesis process to generate an additional state with no slow action transition, but this causes

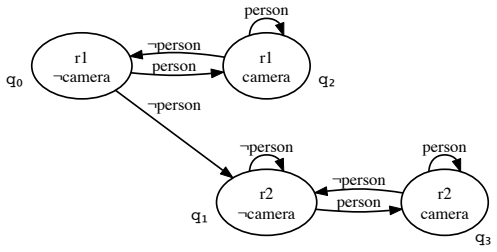


Fig. 5: Synthesized automaton for Example 2 with weakened goal from Section V-A

the system to be unrealizable (because the environment can oscillate the value of π_{person} to keep the system from moving), unless the system goal is weakened as follows:

$\square\diamondsuit(\neg\pi_{person} \Rightarrow \pi_{r_2})$
(Infinitely often, if you are not sensing a person go to r_2)

Fig. 5 depicts the automaton synthesized for this weakened goal. Note that if the robot is in r_1 and a person is sensed, the robot will stay in r_1 until the person is no longer sensed. This could lead to the robot never leaving r_1, which is inconsistent with the original desired behavior.

The need for these extra, unintuitive modifications is undesirable, and the new approach makes them unnecessary by inherently ensuring that fast actions will complete before slow ones. However, the new approach assumes that the environment does not change once the fast action has completed, and ignores any changes until the slower action completes.

B. Consequences for Interpretation of Tense

Example 4 *Consider a similar example, with the addition of a hallway r_{hall} in between r_1 and r_2, and the additional requirement that the camera be turned on if and only if the robot is in the hallway (independent of sensor states):*

$\varphi_{r_1} \wedge \neg\pi_{camera}$
(Robot starts in region r1 with camera turned off)
$\wedge \quad \square(\bigcirc\pi_{camera} \Leftrightarrow \bigcirc\pi_{r_{hall}})$
(Activate camera if and only if you are in the hall)
$\wedge \quad \square\diamondsuit(\varphi_{r_2})$
(Go to r2 infinitely often)

With the approach in [3], this yields an automaton. However, it is unsynthesizable with the new approach, because if the robot tries to move closer to its goal r_2 by entering the hallway, it will have to turn on its camera, but these two events have to occur simultaneously according to the safety $\square(\bigcirc\pi_{camera} \Leftrightarrow \bigcirc\pi_{r_{hall}})$. Since the motion is slower than the action of turning on the camera, these events cannot occur simultaneously under the new execution paradigm, and so the robot can never move between the two rooms, making the specification unsynthesizable.

In the new approach, the desired behavior of activating the camera only after moving into the hallway is achieved by adjusting the relative tenses of the system safety as follows:

$\square(\bigcirc\pi_{camera} \Leftrightarrow \pi_{r_{hall}})$
(Activate camera if and only if you *were* in the hall)

VI. Conclusions and Future Work

This paper describes a challenge of applying formal methods in the physical domain of high-level robot control, namely that of achieving correct continuous behavior from discrete control automata when the low-level controllers have different completion times. A new synthesis algorithm and execution paradigm are presented for the case where robot actions are either *fast* or *slow*. The new approach guarantees immediate reactivity and the safety of intermediate states arising during execution. Future work will generalize the solution to more than two controller execution times (including controllers whose relative completion times are unknown).

Recent work has addressed the question of providing user feedback on a specification that has no implementing controller [12], [13]. A specification may be synthesizable under the original synthesis procedure, but unsynthesizable when checking for safety of continuous executions. In this situation, the user can be alerted to the fact that the relative controller execution times are responsible for the unsynthesizability. The most user-friendly way to present the user with this information is another open question of this work.

VII. Acknowledgements

The authors thank Dr. Nir Piterman for fruitful discussions leading to the writing of this paper.

References

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

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

[3] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[4] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Hybrid Systems*, 2010, pp. 101–110.

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

[6] G. Jing, C. Finucane, V. Raman, and H. Kress-Gazit, "Correct high-level robot control from structured english," in *ICRA*, 2012, pp. 3543–3544.

[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*, 2010, pp. 1988 – 1993.

[8] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *In Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI 06)*. Springer, 2006, pp. 364–380.

[9] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," in *Proceedings of the IEEE*, 2000, pp. 971–984.

[10] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[11] D. C. Conner, A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, vol. 4. IEEE, October 2003, pp. 3546–3551.

[12] V. Raman and H. Kress-Gazit, "Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP," in *CAV*, 2011, pp. 663–668.

[13] V. Raman and H. Kress-Gazit, "Automated feedback for unachievable high-level robot behaviors," in *ICRA*, 2012, pp. 5156–5162.

[14] D. Kozen, "Results on the propositional mu-calculus," *Theoretical Computer Science*, vol. 27, pp. 333–354, 1983.