CYBER-PHYSICAL SYSTEMS

Memory-Loss Resilient Controller Design for Temporal Logic Constraints

M. Abate\textsuperscript{a, b}, W. Stuckey\textsuperscript{c}, L. Lerner\textsuperscript{c}, E. Feron\textsuperscript{d}, S. Coogan\textsuperscript{a, e}

\textsuperscript{a}School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA; \textsuperscript{b}School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA, USA; \textsuperscript{c}Georgia Tech Research Institute, Atlanta, GA, USA; \textsuperscript{d}Department of Electrical Engineering, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia; \textsuperscript{e}School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, GA, USA

ARTICLE HISTORY

Compiled April 27, 2020

ABSTRACT

This paper studies the problem of controlling finite nondeterministic transition systems to satisfy constraints given as linear temporal logic properties. A controller architecture is proposed that maps finite fragments of the state trajectory history to control inputs. This approach avoids the standard controller construction that employs an onboard automaton which is fragile to memory loss or errors. In contrast, the proposed architecture requires storing only a finite sequence of previous system states in memory and is therefore resilient to memory loss. In particular, the system will operate unaltered after such a memory-loss event once the system recollects this finite sequence of system states. A generalized algorithm is outlined for controller synthesis in this manner. Additionally, we demonstrate the construction and implementation of such a memory-loss resilient controller through an experimental demonstration on a differential-drive robot that experiences memory-loss events.

KEYWORDS

Cyber-Physical Systems (CPS), Controller Synthesis, Resilience, Linear Temporal Logic

1. Introduction

When systems are required to satisfy complex temporal objectives (e.g., specifications given in temporal logic), memoryless controllers are insufficient to ensure correct system behavior. A traditional approach for control subject to temporal logic constraints is to store relevant information about the system history in an appropriately constructed memory unit; runtime actions are then chosen as a function of the combined memory and system state [1–3]. This structure is employed, for instance, by controllers
which rely on online automata [4–6]. Such controllers are convenient because they compactly represent relevant system history information into a finite number of memory states. However, they suffer from a need to maintain persistent knowledge over the entire operational life-cycle of the system; if at some time, the current automaton state is lost or altered, assurance cannot be provided going forward. For this reason, it is necessary to design controllers which operate without persistently stored information. Moreover, it is necessary to rectify the disconnect between controller designs which require persistent and precise knowledge of the system history with controller designs which allow for (if not rely on) regular software resets.

In instances where software aging degrades system performance, regular software restarts can be employed to add system robustness [7–10]. This method, referred to in literature as software rejuvenation, increases system resiliency by regularly reinstalling mission objectives from a trusted mission planner [11]. For controlled dynamical systems, methods exist for enforcing set invariance [12] and tracking control objectives [13] in the presence of such memory losses. Methods in this paradigm do not exist, however, for enforcing more complex logical and temporal system objectives.

This paper explores the problem of designing controllers to enforce linear temporal logic (LTL) specifications with no persistently stored information, i.e., no onboard automaton. That way, if at some time the system loses memory, it can restore correct action in a small amount of time. We create such a memory-loss resilient controller through an offline computation in the product-space between the system state-space and the state-space of an automaton corresponding to the operating specification; this allows the system designer to classify the minimum amount of information which is necessary at runtime to assure the system. Synthesis in the product space is standard in formal methods literature [1,2]; it is important to note, however, that while our proposed methodology uses an automaton for synthesis, the automaton itself is not implemented in the resulting controller architecture.

This paper is structured as follows: In Section 3 we contrast two potential controllers designed to enforce LTL specifications; The problem statement of the work is presented in Section 3 and then discussed in Section 4; We solve the problem statement in Section 5, and a generalized procedure is presented for generating controllers with no persistently stored information; The findings of this work are demonstrated in a case study, presented in Section 6.

2. Preliminaries

We model systems as finite-state nondeterministic transition systems, as formalized in Definition 2.1.

Definition 2.1. A system is a tuple \( T = (X, U, f, \Sigma, L) \), where

- \( X \) is a finite set of states,
- \( U \) is a finite set of control inputs,
- \( f : X \times U \rightarrow 2^X \) is a nondeterministic transition relation,
- \( \Sigma \) is a set of labels, and
- \( L : X \rightarrow \Sigma \) is a labeling map.

Such a system could, for instance, be a finite-state abstraction for a set of continuous-time differential equations, or a hybrid dynamical system; see [1,14] for further details. For the remainder of this paper, we use the notation \( L(x_1, \cdots, x_n) \) to denote the
string of labels \( L(x_1, \ldots, x_n) := L(x_1) \cdots L(x_n) \), and we use the symbols \( \Sigma^* \) and \( \Sigma^\omega \) to denote the sets of finite and infinite words over \( \Sigma \), respectively.

We specify mission objectives in linear temporal logic (LTL). LTL combines Boolean logical connectives such as disjunction, conjunction, and implication with temporal operators that permit, for example, specifying that a system criterion \textit{eventually} holds along a system’s execution, is \textit{always} true along a system execution or is true \textit{until} some other condition becomes true. In this way, LTL enables concisely and precisely specifying a wide range of operating behaviors. For a comprehensive discussion on the semantics of LTL, we refer the reader to [15, Chapter 5]. Additionally, we refer the reader to [1, Chapter 9] for a discussion on controller synthesis against LTL specifications.

Formally LTL specifications are interpreted over infinite system executions. To that end, we introduce the semantics of \( \text{LTL}_3 \) (Definition 2.2) in order to assess the finite system runs against LTL properties [16].

**Definition 2.2 (LTL\(_3\) Semantics).** Let \( w \in \Sigma^* \) denote a finite word. The truth value of an \( \text{LTL}_3 \) formula \( \varphi \) with respect to \( w \), denoted \( [w \models \varphi] \), is an element of \( \mathbb{B}_3 = \{ \top, \bot, ? \} \) defined as follows:

\[
[w \models \varphi] = \begin{cases} 
\top & \text{if } \forall \sigma \in \Sigma^\omega : w\sigma \models \varphi \\
\bot & \text{if } \forall \sigma \in \Sigma^\omega : w\sigma \not\models \varphi \\
? & \text{otherwise.}
\end{cases}
\]

Equivalently, the truth value of \( \varphi \) with respect to \( w \) is \textit{true} “\( \top \)” if \( w \) is a good prefix for \( \varphi \), \textit{false} “\( \bot \)” if \( w \) is a bad prefix for \( \varphi \), and \textit{inconclusive} “\( ? \)” otherwise.

Finally, we introduce the automata-based monitoring procedure for \( \text{LTL}_3 \).

**Definition 2.3 (Monitors in \( \text{LTL}_3 \)).** For a given property \( \varphi \) a monitor automaton \( M^\varphi \) is a finite state machine that reads finite words \( w \in \Sigma^* \) and outputs \( [w \models \varphi] \).

For the remainder of this work, we use the tuple representation \( M^\varphi = (\Sigma, Q, q_0, \delta, \lambda) \) to denote the monitor automaton formed corresponding to \( \varphi \), where

- \( \Sigma \) is a set of inputs,
- \( Q \) is a finite set of automaton states,
- \( q_0 \in Q \) is an initial state,
- \( \delta : Q \times \Sigma \to Q \) is a transition relation, and
- \( \lambda : Q \to \mathbb{B}_3 \) is an output function.

Additionally, for \( q \in Q \) and \( \sigma_1, \ldots, \sigma_n \in \Sigma \) we use the notation \( \delta(q, \sigma_1 \cdots \sigma_n) \) to denote the state that \( q \) transitions to after receiving the string of labels \( \sigma_1 \cdots \sigma_n \), defined recursively by \( \delta(q, \sigma_1 \cdots \sigma_n) := \delta(\delta(q, \sigma_1 \cdots \sigma_{n-1}), \sigma_n) \). For a more complete discussion on the construction and implementation of monitors, we refer the reader to [16].

### 3. Invertible Automata and Memory

Suppose a system designer would like to synthesize a controller which enforces the LTL property \( \varphi \) over a run of \( T \). Intuitively, this controller will need access to information about the system history to assess the system’s progress toward the satisfaction of \( \varphi \)
Figure 1.: Two controller designs which assure the system $T = (\mathcal{X}, U, f, \Sigma, L)$ against the LTL specification $\varphi$.

and choose control inputs at runtime. We consider two possible controller architectures to assure the system $T$ against $\varphi$. A block diagram visualization of each architecture is provided in Figure 1.

First, consider a controller that keeps track of the relevant information of the system history by running an onboard automaton $A^\varphi$ (Figure 1a). For a detailed discussion on the generation and implementation of such controllers, we refer the reader to [1]. In this instance, the control input at time $k$ is chosen by an input map $\mu : \mathcal{X} \times Q \rightarrow 2^U$ that receives the current system state $x_k \in \mathcal{X}$ and the current automaton position $q_k \in Q$, where $Q$ denotes the set of states of $A^\varphi$. This architecture is attractive because of its simplicity; knowledge of the current automaton state $q_k$ is sufficient to assess the system’s progress toward the satisfaction of $\varphi$. However, this design suffers from the fact that the automaton block must maintain an uncorrupted feedback loop for the entire operational lifetime of the system; if at some time, the controller loses track of its current position in $A^\varphi$, then all assurances going forward are lost.

As an alternative, consider a controller which chooses inputs by analyzing a finite fragment of the system’s immediate history $x_{k+1-n} \cdots x_k$ (Figure 1b). We use the term history register to denote the internal component which stores such path fragments and the symbol $H_n$ to denote a history register capable of storing $n$ system states. Here, the control input at time $k$ is chosen by an input map $\mu : \mathcal{X}^n \rightarrow 2^U$ that receives the $n$ most recent system states from $H_n$. A simple memoryless state feedback controller $u_k = \mu(x_k)$ can be thought of as a trivial instantiation of this proposed controller architecture, where $n = 1$, though we also allow for more general feedback control laws $u_k = \mu(x_{k-n+1}, \cdots, x_k)$. The addition of the history register to the controller architecture adds system resiliency by removing the need to maintain persistent information over the entire system life-cycle; if, at some time, the history register $H_n$
loses track of the system history, then the system need only wait \( n - 1 \) time steps before the controller has the requisite information to continue choosing safe inputs.

Below we show that for certain LTL properties, knowledge of a finite history fragment is sufficient to correctly identify the system’s progress toward the satisfaction of \( \varphi \). This is the case, in particular, when the monitor automaton which corresponds to \( \varphi \) is invertible, \textit{i.e.,} when there exists an \( n \in \mathbb{N}_{\geq 1} \) such that knowledge of \( n \) previous system states is sufficient to identify the final state of the system run over \( M^\varphi \). We formally define monitor inversion in Definition 3.1 and we provide a sample construction of an invertible monitor automaton in Example 1.

**Definition 3.1 (Invertible Monitor Automaton).** A monitor automaton \( M^\varphi = (\Sigma, Q, q_0, \delta, \lambda) \) is \( n \)-step invertible if, for all \( w \in \Sigma^n \) and all \( v_1, v_2 \in \Sigma^* \), \( \delta(q_0, v_1w) = \delta(q_0, v_2w) \) whenever \( \delta(q_0, v_1w) \neq q_\bot \) and \( \delta(q_0, v_2w) \neq q_\bot \).

**Example 1.** Consider the LTL property \( \varphi_1 = \square(A \rightarrow \bigcirc^2B) \) evaluated over the alphabet \( \Sigma = \{A, B, C\} \). We provide the corresponding monitor automaton \( M^{\varphi_1} \) in Figure 2. As enumerated in Figure 3, knowledge of the two previous system states is sufficient to identify the current monitor state, as each combination of labels in \( \Sigma^2 \) corresponds to one and only one possible monitor position, provided that the previous system trace is known to be safe. Therefore, \( M^{\varphi_1} \) is \( n \)-step invertible for any \( n \geq 2 \). Further, it is not difficult to see that \( M^{\varphi_1} \) is not 1-step invertible, as knowledge of \( L(x_k) \), the current system label, does not by itself uniquely determine the current monitor state. It follows, therefore that a history register based controller which stores two system states will be able to achieve the same assurances as a controller which runs \( M^{\varphi_1} \) onboard. This design, however, has added resiliency and is able to provide

---

**Figure 2.:** Monitor automaton \( M^{\varphi_1} \) for the LTL specification \( \varphi_1 = \square(A \rightarrow \bigcirc^2B) \) evaluated over the alphabet \( \Sigma = \{A, B, C\} \). States \( q_0, \cdots, q_3 \) have output \textit{inconclusive}, “?”, and state \( q_\bot \) has output \textit{false}, “\( \bot \)”.

**Figure 3.:** Knowledge of the two previous system labels is always sufficient to correctly identify the system position inside \( M^{\varphi_1} \). This is exemplified by the fact that each potential two-state system history, \textit{i.e.,} combination of labels in \( \Sigma^2 \), corresponds to a unique current monitor position, provided that the previous system trace is known to be safe.

\[
\begin{array}{|c|c|c|}
\hline
L(x_{k-1}) & L(x_k) & \text{If } [L(x_0 \cdots x_k) \models \varphi_1] \neq \bot \\
\hline
(B \lor C) & (B \lor C) & q_0 \\
(B \lor C) & A & q_1 \\
A & A & q_2 \\
A & (B \lor C) & q_3 \\
\hline
\end{array}
\]
assurance in the presence of memory loss.

Example 1 demonstrates that a history register style controller can provide assurance against \( \varphi \) by inverting the monitor automaton \( M^\varphi \) online and then synthesizing a safe control policy in the product space \( X \times Q \). Note, however, that not all automata are invertible. We demonstrate this assertion in Example 2.

Example 2. Consider the LTL property \( \varphi_2 = \Box(\neg A) \lor (AUB) \) evaluated over the alphabet \( \Sigma = \{A, B, C\} \). We provide the corresponding monitor automaton \( M^{\varphi_2} \) in Figure 4. Knowledge of a finite history fragment \( x_{k-4} \cdots x_k \) will not always be sufficient to identify the system’s progress toward satisfying \( \varphi_2 \). This is exemplified by the fact that if \( L(x_k \cdots x_k) = (A)^n \), then it will not be possible to determine whether the system run over the monitor ends in state \( q_1, q_2 \) or \( q_\top \); this is true regardless of the number of states which are stored. It follows, therefore that \( M^{\varphi_2} \) is not invertible.

For this reason, it is favorable to abandon controller architectures which rely on onboard automata, and instead design controllers which map directly from the system state-space to a set of safe control inputs, with no intermediate automaton inversion. Synthesizing such a controller is equivalent to developing a mapping \( \mu : X^n \to 2^U \) and a safe region of the label-space \( S^b \subseteq X^n \) which is forward-invariant under \( \mu \). We formalize the search for such a mapping as the main problem statement of this work:

**Problem Statement.** Consider a system \( T = (X, U, f, \Sigma, L) \) and an LTL property \( \varphi \). Given \( n \in \mathbb{N}_{\geq 1} \), generate a feedback control policy \( \mu : X^n \to 2^U \) and a region \( S^b \subseteq X^n \) such that

1. \( S^b \) is forward-invariant under \( \mu \), i.e. if \( (x_1, \cdots, x_n) \in S^b \) and \( u \in \mu(x_1, \cdots, x_n) \), then \( (x_2, \cdots, x_{n+1}) \in S^b \), for all \( x_{n+1} \in f(x_n, u) \), and
2. \( S^b \) is safe, i.e. if \( [L(x_0 \cdots x_k) \models \varphi] \neq \bot \), \( (x_{k+1-n}, \cdots, x_k) \in S^b \) and \( u \in \mu(x_{k+1-n}, \cdots, x_k) \), then \( [L(x_0 \cdots x_{k+1}) \models \varphi] \neq \bot \), for all \( x_{k+1} \in f(x_k, u) \).

For the remainder of the work, we call \( \mu_n : X^n \to 2^U \) an \( n \)-step feedback control policy if there exists an \( S^b_n \subseteq X^n \) such that the pair \( (\mu_n, S^b_n) \) solves the problem statement.
4. History Register Based Controllers for Safety

Note that employing an \( n \)-step feedback control policy \( \mu_n \) does not guarantee that the resultant infinite system trajectory satisfies \( \varphi \); rather, employing \( \mu_n \) guarantees that at any given time \( k \geq n \) the current finite system trajectory \([L(x_0, \cdots, x_k)] = \varphi\) \( \neq \bot \). In this sense, one can view \( \mu_n \) as a least-restrictive backup control policy that retains nondeterminism [17]. For certain classes of specifications, however, choosing control inputs according to \( \mu_n \) guarantees infinite-time system satisfaction of \( \varphi \); in this instance, \( \mu_n \) can be employed stand alone. This observation is formalized in the next remark.

**Remark 1.** If \( \varphi \) is an LTL safety property and \((\mu_n, S^b_n)\) solves the problem statement, then the infinite system trace resulting when \( \mu_n \) is applied satisfies \( \varphi \), i.e. \( L(x_0, x_1, \cdots) = \varphi \). This is due to the fact that the satisfaction of a safety property \( \varphi \) can be shown by proving the system trace does not contain a bad prefix for \( \varphi \) [16].

Additionally, note that the assurance capabilities of an \( n \)-step feedback control policy increase as \( n \) increases. This is due to the fact that the ambiguity as to the system’s progress toward the satisfaction of \( \varphi \) is reduced as the number of saved system states increases. In the instance that the monitor automaton \( M^\varphi \) is invertible, however, there exists an intrinsic maximum number of states which will need to be remembered by a controller before there is no ambiguity as to the system’s progress.

We encapsulate this result in Theorem 4.1.

**Theorem 4.1.** Let \( \varphi \) be an LTL property such that the monitor automaton \( M^\varphi \) is \( n \)-step invertible. Let the pair \((\mu_m, S^b_m)\) solve the problem statement where \( \mu_m \) is an \( m \)-state feedback control policy \( m > n \). Then there exists an \( n \)-step feedback control policy \( \mu_n \) with a corresponding region \( S^b_n \), such that if \((x_{k-m}, \cdots, x_k) \in S^b_m\) then

- \((x_{k-n+1}, \cdots, x_k) \in S^b_n\), and
- \( \mu_m(x_{k-m+1}, \cdots, x_k) = \mu_n(x_{k-n+1}, \cdots, x_k) \).

Theorem 4.1 follows immediately from the preceding discussion, and we sketch the proof for this result as follows: if the system remembers \( n \) previous states, then the monitor automaton \( M^\varphi \) can be inverted, to give \( \mu_n \) access to the current state of the system execution over \( M^\varphi \). Storing additional system history states does not give the controller access to more information about the system’s progress.

As shown in Theorem 4.1, the guarantees of a history-register based controller derive from the invertibility of the monitor automaton \( M^\varphi \). Intuitively, a non-invertible monitor automaton will contain two or more symmetrically labeled cycles, so that the system’s progress toward the satisfaction of \( \varphi \) cannot be determined by any history fragment of finite length (see Example 2). As such, one can show that \( M^\varphi \) is \( n \)-step invertible by calculating all transition-cycles of length \( n \) and then checking that no two cycles are symmetric under cyclic permutation. This procedure can be accomplished using the standard graph search algorithms; transition-cycles, for instance, can be identified using a recursive depth first search to produce a modular decomposition of all possible cycles of states in \( M^\varphi \). Such a search will produce a finite set of minimum length sub-cycles, from which all of possible cycles can be characterised.

Importantly, one must only calculate transition-cycles up to length \(|Q|\) before invertibility can be determined. Therefore, the procedure for determining monitor invert-
Proposition 4.2. If the monitor automaton \( \mathcal{M}^\varphi = (\Sigma, Q, q_0, \delta, \lambda) \) is invertible, then there exists an \( n \leq |Q| \) such that \( \mathcal{M}^\varphi \) is \( n \)-step invertible.

Proof. For an LTL property \( \varphi \), assume the monitor automaton \( \mathcal{M}^\varphi \) is invertible. Choose \( \sigma \in \Sigma \). As a consequence of the fact that \( \mathcal{M}^\varphi \) is invertible, there cannot be a 2-state cycle in \( \mathcal{M}^\varphi \) with edges labeled \( \sigma \), i.e. there cannot be two states \( q_1, q_2 \in Q \) such that \( \delta(q_1, \sigma) = q_2 \) and \( \delta(q_2, \sigma) = q_1 \). This assertion extends to cycles of any length and, thus, there must exist a unique \( q_\sigma \in Q \) (possibly with \( q_\sigma = q_\perp \)) such that the following two conditions hold:

1. \( \delta(q_\sigma, \sigma) = q_\sigma \), and
2. for all \( q \in Q \) and \( y \geq |Q| \) we have \( \delta(q, \sigma^y) \in \{q_\sigma, q_\perp\} \).

Moving recursively, it can be shown that for every \( w \in \Sigma^* \) such that \( |w| \geq |Q| \) there exists a unique \( q_w \in Q \) (possibly with \( q_w = q_\perp \)) such that

1. \( \delta(q_w, w) = q_w \), and
2. for all \( q \in Q \) we have \( \delta(q, w) \in \{q_w, q_\perp\} \).

Therefore if \( \mathcal{M}^\varphi \) is invertible, then \( \mathcal{M}^\varphi \) is \( n \)-step invertible. This completes the proof. \( \square \)

5. Computing an \( n \)-step feedback control policy

Consider a system \( T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L) \). Given \( n \geq 1 \) and LTL specification \( \varphi \), we aim to develop an \( n \)-step feedback control policy \( \mu : \mathcal{X}^n \rightarrow \mathcal{U} \) which assures \( T \) against \( \varphi \) in some region \( S^b \subseteq \mathcal{X}^n \). Here, \( n \) could be chosen to either ensure the \( n \)-step invertibility of \( \mathcal{M}^\varphi \), if \( \mathcal{M}^\varphi \) is invertible, or to meet some other specification on the system design; in the latter case, the safe zone \( S^b \) may be smaller, however the system will recover faster in the event of memory loss. We solve the problem statement by applying the following steps:

1. We first form a product system \( P_n = T \otimes_n \mathcal{M}^\varphi \) which encodes the linked progression of the system and monitor automaton.
2. We then define an equivalence relation \( \sim \) that equates two product states if these states are both safe and observationally equivalent in a history register. We use the symbol \( P_n/\sim \) to denote the quotient of \( P_n \) under observational equivalence, which we formalize later in the work.
3. Finally, we compute the largest forward-invariant region of the quotient transition system \( S^b \subseteq P_n/\sim \), and a corresponding safe control policy \( \mu : S^b \rightarrow 2^\mathcal{U} \).

It is important to note that while our proposed methodology uses a monitor automaton to construct an \( n \)-step feedback control policy, the monitor is not used at runtime. Rather, control inputs are chosen online by the preformed policy \( \mu \).

In order to synthesize a controller capable assuring \( T \) against an LTL property \( \varphi \), it is necessary to perform analyses in the product space of the system and the objective.

Definition 5.1. The \( n \)-th controlled monitor product automaton \( P_n = T \otimes_n \mathcal{M}^\varphi \) of a system \( T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L) \) and a monitor automaton \( \mathcal{M}^\varphi = (\Sigma, Q, q_0, \delta, \lambda) \) is defined to be \( P_n = (\mathcal{X}, \mathcal{U}, f, \mathcal{O}, \sigma) \), where
\( \mathcal{X} \subseteq X^n \times Q \) is a set of states, where

\[
\mathcal{X} = \{ (x_{p+1}, \ldots, x_{p+n}, q) \in X^n \times Q \mid \exists x_0, \ldots, x_p \in X, \exists u_0, \ldots, u_{p+n-1} \in U \text{ such that } q = \delta(q_0, L(x_0, \ldots, x_{p+n})) \text{ and } x_{i+1} \in f(x_i, u_i) \text{ for all } 0 \leq i \leq p + n - 1 \}
\]

- \( \mathcal{X} \) is a set of states,
- \( f: X \times U \rightarrow 2^X \) is the transition map, where \( (x_2, \ldots, x_{n+1}, q') \in f(x_1, \ldots, x_n, q, u) \) if and only if \( x_{n+1} \in f(x_n, u) \) and \( q' = \delta(q, L(x_{n+1})) \),
- \( O = X^n \cup \{ x_\perp \} \) is a set of observations, and
- \( \sigma: \mathcal{X} \rightarrow O \) is the observation map, where

\[
\sigma(x_1, \ldots, x_n, q) = \begin{cases} x_\perp & \lambda(q) = \perp \\ (x_1, \ldots, x_n) & \text{otherwise.} \end{cases}
\]

Each state of the product automaton \( P_n = T \otimes_n M^\nu \) denotes a possible system history fragment paired with a possible monitor state; similarly, a run over the product automaton encodes a system trajectory and its corresponding monitor run. We therefore compute \( \mathcal{X} \) by first forming \( \mathcal{X} \times Q \) and then removing the states which do not appear in valid system-monitor trajectories.

Note that the observation \( \sigma(s) \) of a product state \( s \in \mathcal{X} \) is equal to \( x_\perp \) if and only if the system trace up until this point is a bad prefix for the mission objective. Therefore system safety is only achievable in the instance that the system trace over the product automata \( s_0, s_1, s_2, \ldots \in \mathcal{X}^\nu \) does not contain a state with output false, i.e. \( \sigma(s_i) \neq x_\perp \) for all \( i \). For this reason, we allow the controller to choose control inputs based on the assumption that the system has acted safely up until the current time, i.e., at time \( k \), the controller assumes \( \sigma(s_k) \neq x_\perp \), where \( s_k \) denotes the unknown current system state in the product automaton. Under this assumption, the observation \( \sigma(s_k) \) is equal to the total amount of information available to the controller, when the system and monitor is in state \( s_k \in \mathcal{X} \). Moreover, we can design controllers to assure the original system by developing policies which assure the product system given only the observation of the current state. We formalize this assertion as Proposition 5.2.

**Proposition 5.2.** For \( \mu: X^n \rightarrow 2^U \) and \( S^b \subseteq X^n \) define

\[
\overline{S}^b := \{ s \in \mathcal{X} \mid \sigma(s) \in S^b \}
\]

and

\[
N(s) := \bigcup_{u \in \mu(\sigma(s))} f(s, u)
\]

so that \( \overline{S}^b \subseteq \mathcal{X} \) is the set of product states whose observations are contained inside \( S^b \) and \( N: \mathcal{X} \rightarrow 2^\mathcal{X} \) is a function which takes a product state \( s \in \mathcal{X} \) and returns the set of product states reachable from \( s \) when the input is chosen by \( \mu \). If \( N(s) \subseteq \overline{S}^b \) for all \( s \in \overline{S}^b \) then the pair \( (\mu, S^b) \) solves the problem statement.
Proof. Assume \( N(s) \subseteq \overline{S}^b \) for all \( s \in \overline{S}^b \). Then \( x_\perp \not\in \overline{S}^b \) and \( \overline{S}^b \) is forward invariant on \( P_n \) when control actions are chosen using \( \mu \). Further, \( S^b \) is forward invariant on \( T \) when control actions are chosen using \( \mu \), and all system runs constrained to \( S^b \) are not bad prefixes for \( \varphi \). Therefore \((\mu, S^b)\) solves the problem statement. \( \square \)

Next we equate the product states which are indistinguishable under observation. We refer to this procedure as taking the quotient of \( P_n \) under observational equivalence, as formalize in Definitions 5.3 and 5.4.

Definition 5.3. Let \( P_n = (\overline{X}, \mathcal{U}, f, \overline{O}, \overline{\sigma}) \) be a product system. Two states \( s_1, s_2 \in \overline{X} \) are observationally equivalent (written \( s_1 \sim s_2 \)) if and only if \( \overline{\sigma}(s_1) = \overline{\sigma}(s_2) \).

Definition 5.4. Given a system \( T = (X, \mathcal{U}, f, \Sigma, L) \) and a LTL property \( \varphi \), let \( P_n = (\overline{X}, \mathcal{U}, \overline{f}, \overline{O}, \overline{\sigma}) \) be the product \( P_n = T \otimes_n \mathcal{M}^\varphi \). The automaton quotient is defined to be \( P_n/\sim = (X_\sim, X_\perp, \mathcal{U}, f_\sim) \), where

- \( X_\sim = X^n \cup \{ x_\perp \} \) is a set of states.
- \( X_\perp \) is a bad state.
- \( \mathcal{U} \) is a set of control inputs, and
- \( f_\sim : X_\sim \times \mathcal{U} \rightarrow 2^{X_\sim} \) is a nondeterministic transition relation, \( (x_1, \cdots, x_{n+1}) \in f_\sim(x_1, \cdots, x_n, u) \) if and only if there exists \( s_1, s_2 \in \overline{X} \) such that \( s_2 \in \overline{f}(s_1, u), (x_1, \cdots, x_n) = o(s_1) \), and \( (x_2, \cdots, x_{n+1}) = o(s_2) \).

Recall that given a product state \( s \in \overline{X} \), the observation \( \overline{\sigma}(s) \) encodes the information which will be available to the controller in the instance that the system run over the product automaton ends in state \( s \). As \( x_\perp \in \overline{O} \) encodes the set of product states which are unsafe, system safety is ensured by choosing control inputs such that the system never enters \( x_\perp \in X_\sim \) in the quotient automaton. To that end, we strictly define the region from which assurance can be guaranteed; this region is referred to as a winning component of \( P_n/\sim \).

Definition 5.5. Let \( P_n/\sim = (X_\sim, x_\perp, \mathcal{U}, f_\sim) \) be the quotient of \( P_n = T \otimes_n \mathcal{M}^\varphi \) under product observational equivalence. A winning component \( S^b \subset X_\sim \) is any set of quotient states such that for each history fragment \( (x_1, \cdots, x_n) \in S^b \) there exists an infinite sequence of inputs such that the system run over \( P_n/\sim \) starting from \( (x_1, \cdots, x_n) \) never enters \( x_\perp \). A winning policy is any mapping \( \mu : X_\sim \rightarrow 2^\mathcal{U} \) such that if \( (x_1, \cdots, x_n) \in S^b \), then \( f_\sim(x_1, \cdots, x_n, u) \subseteq S^b \), for all \( u \in \mu(x_1, \cdots, x_n) \).

Numerous methods exist for identifying controlled invariant regions in automata [18,19]. Here, we include one such algorithm, specific to our application (Algorithm 1); \( \text{Win}(P_n/\sim) \) takes an arbitrary quotient system \( P_n/\sim \) and returns a maximal set of winning states \( S^b \subset X_\sim \). The corresponding winning policy \( \mu \) can then be calculated according to \( \mu(x) = \{ u \in \mathcal{U} \mid x \in S^b \Rightarrow f_\sim(x, u) \subseteq S^b \} \).

These tools are sufficient to solve the problem statement. Therefore, we now formalize the main result of the work (Theorem 5.6) and then demonstrate this result (Example 3).

Theorem 5.6. Given a transition system, \( T \), and an LTL property \( \varphi \), let \( P_n/\sim = (X_\sim, x_\perp, \mathcal{U}, f_\sim) \) denote the quotient of \( P_n = T \otimes_n \mathcal{M}^\varphi \) under product observational equivalence. Let \( S^b \subseteq X_\sim \) be a winning component of \( P_n/\sim \), with a corresponding winning policy \( \mu \). Then the pair \((\mu, S^b)\) solves the Problem Statement.

Proof. By definition, \( S^b \subseteq X^n \) is forward-invariant under the control-policy \( \mu \). Con-
consider a finite run of the system \( x_0, \ldots, x_k \), such that \( [L(x_0, \cdots, x_k) \models \varphi] \neq \bot \); by definition, the system run over the monitor automaton \( M^\varphi \) is guaranteed to end in a monitor state \( q_k \neq q_\bot \). We can equivalently consider the system run over the product automaton \( P_n = T \otimes_n M^\varphi \), which will end in a state \( s_k = (x_{k+1-n}, \ldots, x_k, q_k) \in \mathcal{X} \) where \( \pi(s_k) = \hat{s}_k \neq x_\bot \). Now consider the state \( \hat{s}_k \in \mathcal{X} \) in the automaton quotient \( P_n/\sim \), and assume \( \hat{s}_k \in S^b \). If the control input \( u_k \in \mu(\hat{s}_k) \) is guaranteed to take \( \hat{s}_k \) to a state \( \hat{s}_{k+1} \in S^b \), where \( \hat{s}_{k+1} \neq x_\bot \), then \( u_k \) is guaranteed to take \( s_k \) to a state \( s_{k+1} = (x_{k+2-n}, \ldots, x_{k+1}, q_{k+1}) \) such that \( q_{k+1} \neq q_\bot \). Equivalently, \( [L(x_0, \cdots, x_{k+1}) \models \varphi] \neq \bot \) for all \( x_{k+1} \in f(x_k, u_k) \). Therefore, \( S^b \) is safe.

**Example 3.** Consider a system whose dynamics are encoded in the transition system \( T = (X, U, f, \Sigma, L) \), where

- \( X = \{x_1, x_2, x_3\} \) is a set of system states,
- \( U = \{u_1, u_2\} \) is a set of control inputs,
- \( f : X \times U \rightarrow 2^X \) is a transition relation, as given in Figure 5a,
- \( \Sigma = \{A, B, C\} \) is a set of labels, and
- \( L : X \rightarrow \Sigma \) is a labeling map, as given in Figure 5a.

We aim to enforce the LTL property \( \varphi = \Box(A \rightarrow \Box^2 B) \) (studied previously in Example 1) over a run of \( T \), using a history-register style controller with \( n = 1 \) memorized system state. We synthesize such a controller by forming the product automaton \( P_1 = T \otimes_1 M^\varphi \) (Figure 5b), and then searching for the largest forward-invariant region \( S^b_1 \) in the quotient automaton \( P_1/\sim \) (Figure 5c). In this case, assurance can be provided over \( S^b_1 = \{x_1, x_2\} \subset X \), where control inputs are chosen according to

\[
\mu_1(x) = \begin{cases} 
\{u_2\} & x = x_1 \\
\{u_1\} & x = x_2 \\
? & x = x_3,
\end{cases}
\]

where the symbol “??” denotes the control output in the case that correct performance cannot be assured.

We now repeat this procedure to create the input map \( \mu_2 : X^2 \rightarrow 2^U \). We synthesize this control policy by forming the two-state product automaton \( P_2 = T \otimes_2 M^\varphi \) and
then searching for the largest forward-invariant region $S_2^h$ in the automata quotient $P_2/\sim$ (Figure 5d). In the case of this example, assurance can be provided over the region $S_2^h \subset X^2$ where control inputs are chosen according to

$$
\mu_2(s) = \begin{cases} 
\{u_1\} & s = (x_1, x_2) \\
\{u_2\} & s \in \{(x_2, x_1), (x_3, x_3)\} \\
\{u_1, u_2\} & s \in \{(x_2, x_2), (x_2, x_3), (x_3, x_1)\} \\
\emptyset & s = (x_1, x_3) 
\end{cases}
$$

Then

$$
S_2^h = \{(x_1, x_2), (x_2, x_1), (x_3, x_3), (x_2, x_2), (x_2, x_3), (x_3, x_1)\}.
$$

Note that a controller which chooses inputs according to $\mu_2$ will have significant
advantages over a controller which chooses inputs according to $\mu_1$. For instance, at a time $k$, a controller which chooses inputs according to $\mu_1$ will be able to provide assurance if and only if the current system state $x_k \in \{x_1, x_2\}$; whereas, a controller which chooses inputs according to $\mu_2$ will be able to provide assurance when $x_k = x_3$, provided that the previous system state is known to be $x_{k-1} \in \{x_2, x_3\}$. Additionally, in some cases $\mu_2$ allows the system designer to choose which control input is applied. For instance, if, at a time $k$, the current system state $x_k = x_1$, then $\mu_1$ will apply control input $u_k = u_2$ to $T$; whereas, in the same situation, the control policy $\mu_2$ will allow the system designer to choose between the control input $u_k = u_1$ and $u_k = u_2$, provided that the previous system state is known to be equal to $x_{k-1} = x_3$. Finally, recall from Example 1 that $\mathcal{M}^{\varphi}$ is invertible with two memorized system. Therefore, increasing the memory capabilities of the controller beyond $n = 2$ memorized system states will not increase the number of instances when assurance is guaranteed, as shown in Theorem 4.1.

6. Experimental Demonstration

We now present a case study demonstrating the findings of the work. In this study we design three different history-register based controllers to assure a differential drive robot against an LTL safety property. These controllers are generated using MATLAB 2020a and implemented on an actual robotic testbed.

6.1. Controller Synthesis

Consider an autonomous vehicle which must refuel periodically, while also avoiding obstacles. We abstract the vehicle dynamics in a 200 state nondeterministic transition system $T$, representative of a discrete-time discrete-state unicycle model. Here, each state of $T$ denotes a unique position $(x, y) \in \{1, \ldots, 5\} \times \{1, \ldots, 5\}$ and rotation $\theta \in \{0, \frac{\pi}{4}, \ldots, \frac{7\pi}{4}\}$. The control inputs to the system are linear velocity $v \in \{0, 1\}$ and angular velocity $\omega \in \{0, \pm\frac{\pi}{4}\}$. Nondeterminism in the system arises when the control input $(v, \omega) = (1, \pm\frac{\pi}{4})$, i.e. when the system intends to drive forward and turn during the same time-step. In this case the system will nondeterministically enter either the state arising from moving forward one unit and then rotating, or the state arising from rotating first and then moving forward one unit. This problem setting is shown in Figure 6.

We stipulate that the vehicle must spend 2 consecutive time-steps in a refueling region every 8 time-steps and always avoid obstacles. This mission objective is captured in the LTL safety specification

$$\varphi = \square(\neg \text{Obs}) \land \square(v_{i=0}^{7} \Box^{i} (\text{Ref} \land \Box \text{Ref}))$$

where refueling states are labeled $\text{Ref}$ and obstacle states are labeled $\text{Obs}$. We use the symbol $\varnothing$ to represent the system label when the vehicle is neither refueling or at an obstacle position. Therefore, we evaluate $\varphi$ over the alphabet $\Sigma = \{\text{Obs}, \text{Ref}, \varnothing\}$; the labeling of $T$ over $\Sigma$ is shown in Figure 6a. The monitor automaton $\mathcal{M}^{\varphi}$ was formed using Spot 2.0 [20]; the resultant automaton is composed of 16 monitor states, 15 of which output inconclusive and 1 of which is a false trap-state. A graph search conducted using MATLAB 2020a shows $\mathcal{M}^{\varphi}$ to be 7-step invertible; this search is conducted by computing all words $w \in \Sigma^7$ and then showing that $\delta(q_0, v_1 w) = \delta(q_0, v_2 w)$.
Two-dimensional projection of system state space. Refueling (Ref) states are shown in blue, obstacle (Obs) states are shown in pink, and the remaining states (∅) are shown in yellow. Vehicle states $x_1 = (1, 5, 7\pi/4)$ and $x_2 = (5, 2, \pi/2)$ are shown as purple triangles, where $L(x_1) = \text{Ref}$ and $L(x_2) = \varnothing$.

(b) Visualisation of nondeterminism in the system model. When beginning at an initial state $(1, 1, \pi/4)$, shown in purple, and applying the control input $(v, \omega) = (1, -\pi/4)$ the system will nondeterministically enter either of the states shown in light pink: $(2, 2, 0)$ which is the state arising from moving forward one unit and then rotating, or $(2, 1, 0)$ which is the state arising from rotating first and then moving forward one unit.

Figure 6.: Problem setting. Figure 6a shows the labeling of the system state space, and Figure 6b describes the nondeterminic control actions.

all $v_1, v_2 \in \Sigma^*$ whenever $\delta(q_0, v_1w) \neq q_\perp$ and $\delta(q_0, v_2w) \neq q_\perp$. Therefore, a history register based controller which stores seven previous system states will provide the same guarantees as a traditional automaton based controller. In the remainder of this section, we explore the guarantees that are possible with even fewer memory states.

We now consider three alternative controllers constructed to assure $T$ against $\varphi$ where these controllers are required to make runtime decisions based on $n \in \{1, 2, 3\}$ memorized system states. These controllers are formed according to the procedures laid out in Section 5, i.e., we identify a controlled invariant region of the quotient system $P_n/\sim$ and the corresponding winning policy $\mu_n$. In general, such controllers will not be able to guarantee $\varphi$ from all initial states. Intuitively, this is because $\varphi$ requires refueling only every 8 time-steps, but a controller with limited memory will need to act conservatively and guarantee refueling within its memory horizon as it cannot know of refueling events further in the past. However, if the system is initialized within the winning invariant region, then they controller will ensure that $\varphi$ is satisfied along the infinite trace of the system execution.

First, a control policy $\mu_1 : \mathcal{X} \rightarrow 2^U$ is formed to provide assurance when $n = 1$ system state is available to the controller. Note $\mu_1$ is memory-less and makes control decisions based solely on the current vehicle state. $\mu_1$ has a corresponding region $S^b_1 \subset \mathcal{X}$, and is able to provide assurance over $|S^b_1| = 62$ of the 200 system states. Here, $S^b_1$ is comprised of every refueling state, as well as the states which could reach a refueling state in one time step; $\mu_1$ then either brings the vehicle to the refueling region or forces the vehicle to remain inside the fueling region. At any other state outside $S^b_1$, a memoryless controller is unable to guarantee satisfaction of $\varphi$ since it is unable to reach a refueling state in the next step and has no memory of the last refueling event.

Next, a control policy $\mu_2 : \mathcal{X}^2 \rightarrow 2^U$ is formed to provide assurance when $n = 2$ system states are available to the controller. $\mu_2$ has corresponding region $S^b_2 \subset \mathcal{X}^2$, where assurance can be provided over $|S^b_2| = 369$ possible two-state system histories.
The controller $\mu_2$ allows the vehicle to enter 123 of the possible 200 states without violating the system specification.

Finally, a control policy $\mu_3 : X^3 \to 2^U$ is formed to provide assurance with $n = 3$ stored history states. The controller $\mu_3$ has a corresponding safe region $S^3_{b} \subset X^3$ and is able to provide assurance over $|S^3_{b}| = 2310$ possible three-state system histories. This controller allows the vehicle to enter 134 of the possible 200 states without violating the system specification. Note that 64 of the 200 system states have label $\text{Obs}$ and are inherently inadmissible; therefore, $\mu_3$ allows the system to enter all but 2 admissible system states and is therefore almost as effective as the least conservative policy that requires seven history states. Provided that the initial state of the system and its history register is within $S^3_{b}$, the controller $\mu_3$ ensures satisfaction of $\varphi$. Additionally, for certain three-state system histories, $\mu_3$ allows the nondeterministic control input $(v, \omega) = (1, \pm \pi/4)$, which is not the case for $\mu_1$ and $\mu_2$. This indicates that the controller has an assurance plan for both possible states at the next time-step and can allow the nondeterministic action without risk.

A simulated system trajectory under $\mu_3$ is shown in Figure 7; this simulation is conducted in MATLAB 2020a. In the following section, we provide an experimental demonstration of $\mu_3$ on a robotic testbed.

### 6.2. Implementing a Memory-Loss Resilient Controller on a Differential Drive Robot

The three-state history register based controller $\mu_3$ is implemented on a PiBorg model MonsterBorg car (Figure 8). The robot conforms to differential drive dynamics, and is controlled onboard by a Raspberry Pi. Localization is performed using APRILTAGS_ROS.
By choosing control inputs according to $\mu_3$, the PiBorg car is guaranteed to operate continuously without violating $\varphi$, even in the event of memory loss. In the experiment, the vehicle operated continuously for 3 minutes, and was subjected to two artificial memory-loss events. By making intermediate decisions to avoid obstacles, the controller was, in both instances, able to regain a full history register; at this point, $\mu_3$ was reemployed. During the three time steps during which the system is recollecting states in its history register, there is an inevitable risk from the memory loss event that the system will falsify $\varphi$, i.e., a bad prefix for $\varphi$ is encountered, but after these three steps, if such a prefix is not encountered, then the system regains its guarantee of satisfying $\varphi$ for all future time. The resulting finite system trace over $\Sigma$ in this example was not a bad prefix for $\varphi$ and thus the system safely recovered from the memory-loss events. A video of this experiment is available at https://www.youtube.com/watch?v=bn9fCWTrfpY.

7. Acknowledgements

The authors wish to thank Elizabeth Prucka for her work with testbed development.

8. Conclusion

This paper proposes a controller architecture that analyzes a finite fragment of the system history at each timestep in order to enforce a linear temporal logic property online. Unlike standard methods, the resulting controller does not rely on an onboard automaton, and therefore can withstand memory loss without compromising system safety. We demonstrate the implementation of such a memory-loss resilient controller through an experimental demonstration on a differential-drive robot.

References


