Synthesis of Universal Safety Controllers

Bernd Finkbeiner[1] , Niklas Metzger[1] , Satya Prakash Nayak[2] , and
Anne-Kathrin Schmuck[2]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{finkbeiner,niklas.metzger}@cispa.de
[2] Max Planck Institute for Software Systems, Kaiserslautern, Germany
{sanayak,akschmuck}@mpi-sws.org

Abstract. The goal of *logical controller synthesis* is to automatically
compute a control strategy that regulates the discrete, event-driven be-
havior of a given plant s.t. a temporal logic specification holds over all
remaining traces. Standard approaches to this problem construct a two-
player game by composing a given complete plant model and the logical
specification and applying standard algorithmic techniques to extract
a control strategy. However, due to the often enormous state space of
a complete plant model, this process can become computationally in-
feasible. In this paper, we introduce a novel synthesis approach that
constructs a *universal controller* derived solely from the game obtained
by the standard translation of the logical specification. The universal
controller's moves are annotated with *prophecies* – predictions about
the plant's behavior that ensure the move is safe. By evaluating these
prophecies, the universal controller can be adapted to any plant over
which the synthesis problem is realizable. This approach offers several
key benefits, including enhanced scalability with respect to the plant's
size, adaptability to changes in the plant, and improved explainability of
the resulting control strategy. We also present encouraging experimental
results obtained with our prototype tool, UNICON.

1 Introduction

Reactive synthesis, in the tradition of Church's Problem [15], and supervisory
control, in the tradition of Ramadge and Wonham [36], both aim at the auto-
matic construction of systems that are guaranteed to satisfy their formal speci-
fications[3]. Reactive synthesis is based on a logical specification, typically given
as a formula of a temporal logic; supervisory control is based on an open system
model, known as the *plant*. In practical applications, we often encounter a combi-
nation of both views, where, for example, the behavioral objectives of a robot are
specified as a temporal formula, while its surroundings, such as walls, humans,
and other robots, as well as the physics of their interaction, are described by a
plant model. This has lead to approaches which automatically abstract physi-
cal behavioral constrains into a plant automaton [38,8,40] or model the plant

---

[3] See [19,37] on an in-depth discussion of their resulting algorithmic differences.

directly as an event-based automaton [32,39], which is then composed with the finite automaton derived from the logical specification. The resulting ω-regular game can then be solved by standard algorithmic techniques from reactive synthesis [3,28,29]. This approach is, however, severely limited in its scalability. The necessity to include *all possible* plant behaviors into the plant model to achieve a correct-by-design solution leads to a huge state space of the plant automaton, which makes solving the resulting game computationally impractical. The situation is even worse if the plant model is not fully known or changes over time, for example as the robot receives new mapping data, which currently requires a complete re-synthesis.

An encouraging observation is, however, that, in practice, it is often not necessary to analyze the interaction of the controller and the plant in great detail. In supervisory control, it has been observed that, if certain conditions are met, the plant already guarantees the desired behavior entirely on its own, independently of the controller [26]. Conversely, benchmarks in reactive synthesis, where there is no explicit plant model, often capture the relevant properties of the plant with just a small set of assumption formulas [27].

Based on this observation, this paper presents a novel synthesis approach that decouples the synthesis of a *universal controller*, which we construct purely based on the temporal formula, from the *adaptation* of the universal controller to the specific plant model. The two phases are connected by an annotation of the output decisions in the universal controller with a condition about the plant behavior, which we call the *prophecy* of the output: the controller only chooses the output if the plant satisfies the prophecy. The main advantage of the new approach is the improved scalability due to the fact that the universal controller is constructed *independently of the plant*. We illustrate the idea with a toy example, which will also serve as the running example of the paper.

Example 1. Our setting consists of the controller c, the plant p, and the external environment e. We assume that each of these three processes has a boolean output proposition: o_c, o_p, and o_e, respectively. At each time step, the processes chose the values for their respective output propositions based on the history of all previous outputs. The temporal specification is given as the LTL formula

$$\varphi := (o_c \leftrightarrow o_p) \; W \; (\bigcirc \neg o_e). \tag{1}$$

The formula requires that the controller matches the output of the plant as long as the environment sets its output to true. As the controller cannot observe the plant output before setting its own output, it is impossible to build a correct controller independently of the plant: unless the environment sets its output to false, the controller must know the output produced by the plant one time step in advance. We express this information with the prophecies $\langle o_p \rangle$, indicating that the plant will output true, and $\langle \neg o_p \rangle$, indicating that the plant will output false. The universal controller then reacts to $\langle o_p \rangle$ by setting o_c to true and to $\langle \neg o_p \rangle$ by setting o_c to false.

Contribution. In the remainder of the paper, we formally introduce the concepts of universal controllers and prophecies. We provide an algorithm to synthesize universal controllers from safety specifications, with a finite representation of the prophecies as tree automata. Furthermore, we show that universal controllers are also *most permissive*, capturing all controller strategies that are correct for any realizable plant model. We present an exploration algorithm to obtain a controller from the universal controller for a specific plant model. We show that the algorithm is sound and complete for safety specifications. Finally, we provide experimental results based on our prototype tool, UNICON [24], showing that our approach scales better than the standard method on synthesis problems with large plant models.

Related Work. One key feature of the universal controller is its *permissiveness*: it captures all control strategies that are correct for any realizable plant. The concept of permissiveness is well-established in the field of supervisory control, where it is *the* key feature of the computed supervisor [14]. However, here permissiveness is understood w.r.t. enabling all behaviors that a *single plant* fulfills. In the same spirit, permissive solutions have also been considered in the field of reactive synthesis [9,12,25], however mostly to ensure variability of resulting implementations. Recently, permissiveness for control applications of reactive synthesis has been considered [31,5], but also w.r.t. one fixed plant model.

Another line of work focuses on synthesizing control strategies that are correct for a particular set of plant models. For example, *dominant strategies* [16,23] are correct for all realizable plants, whereas *admissible strategies* [10,7] are correct for a maximal set of realizable plants. Other works [4,13,21,22] restrict plant models with reasonable assumptions and synthesize a control strategy that is correct for this restricted set of plants. Nevertheless, all these works are restricted by computing *a single control strategy* that must be correct for the entire set of considered plants, whereas our approach computes a universal controller that can be adapted to *different control strategies* depending on the plant model.

Finally, the use of prophecy variables is a common proof technique to make information about future events accessible [2]. A recent work by Beutner and Finkbeiner [11] uses such prophecies to provide a complete algorithm for hyperproperty verification. Inspired by this work, we use prophecies to guess the future behavior of the plant in order to synthesize a universal controller.

2 Preliminaries

For every set X, we write X^* and X^ω to denote the sets of finite and respectively infinite, sequences of elements of X, and let $X^\infty = X^* \cup X^\omega$. For $\rho \in X^\infty$, we denote $|\rho| \in \mathbb{N} \cup \{\infty\}$ the length of ρ, and define $dom(\rho) \coloneqq \{0, 1, \ldots, |\rho| - 1\}$. For $\rho = x_0 x_1 \cdots \in X^\infty$ and $i, j \in dom(\rho)$ with $i \leq j$, we define $\rho[i] \coloneqq x_i$ and $\rho[i, j] \coloneqq x_i \cdots x_j$. Moreover, $last(\rho)$ denotes the last element of ρ. For $\rho \in X^\infty$ and some set Y, we write $\rho \downarrow_Y$ to denote the sequence ρ' with $\rho'[i] = \rho[i] \cap Y$.

System Architectures. In this paper, we focus on systems with three processes, $\mathrm{Proc} = \{\mathsf{c}, \mathsf{p}, \mathsf{e}\}$, i.e., a controller, a plant, and an environment. For the rest of the paper, we fix a set of atomic propositions $\mathtt{AP} = O_\mathsf{e} \uplus O_\mathsf{c} \uplus O_\mathsf{p}$ partitioned into the output propositions of the processes. This defines an architecture $\mathtt{Arc} = (I_\mathsf{e}, O_\mathsf{e}, I_\mathsf{c}, O_\mathsf{c}, I_\mathsf{p}, O_\mathsf{p})$ over $\mathtt{AP}$ where the input propositions of a process $i \in \mathrm{Proc}$ are the union of the output propositions of all other processes, i.e., $I_i = \bigcup_{j \neq i} O_j$.

Traces and Specifications. We fix alphabet $\Sigma = 2^\mathtt{AP}$ to the power set of the atomic propositions $\mathtt{AP}$. A *trace* over Σ is a sequence $\gamma \in \Sigma^\omega$. A *specification* φ specifies some restrictions on the traces. We write $\gamma \vDash \varphi$ to denote that the trace γ satisfies the specification φ. The language $\mathcal{L}(\varphi)$ of a specification φ represents the set of traces that satisfy φ.

LTL. Linear-time temporal logic (LTL) [35] is a specification language for linear-time properties. LTL specifications over atomic propositions $\mathtt{AP}$ are given by $\varphi, \psi ::= \alpha \in \mathtt{AP} \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \bigcirc\varphi \mid \varphi\ U\ \psi \mid \varphi\ W\ \psi \mid \Diamond\varphi \mid \Box\varphi$. A trace is evaluated against an LTL specification φ in the ususal way. We refer to [6, Chapter 5.1.2] for the detailed semantics.

Process Strategies. We model a strategy for process i as a Moore machine $\mathcal{M} = (S, s_0, \tau, o)$ over inputs I_i and outputs O_i, consisting of a finite set of states S, an initial state $s_0 \in S$, a transition function $\tau : S \times 2^{I_i} \to S$ over inputs, and an output labeling function $o : S \to 2^{O_i}$. For a finite input sequence $\gamma = \alpha_0\alpha_1 \cdots \alpha_{k-1} \in (2^{I_i})^*$, $\mathcal{M}$ produces a finite path $s_0 s_1 \cdots s_k$ and an output sequence $o(s_0)o(s_1)\cdots o(s_k) \in (2^{O_i})^*$ such that $\tau(s_j, \alpha_j) = s_{j+1}$. We write $out(\mathcal{M}, \gamma)$ to denote the last output produced, i.e., $o(s_k)$. Similarly, for an infinite input sequence $\gamma \in (2^{I_i})^\omega$, $\mathcal{M}$ produces an infinite path $s_0 s_1 \cdots$ and an infinite output sequence $o(s_0)o(s_1)\cdots \in (2^{O_i})^\omega$. We write $Traces(\mathcal{M})$ to denote the set of all traces $\gamma \in \Sigma^\omega$ such that for input sequence $\gamma \downarrow_{I_i}$, $\mathcal{M}$ produces the output sequence $\gamma \downarrow_{O_i}$. We say that $\mathcal{M}$ satisfies a specification φ, denoted $\mathcal{M} \models \varphi$, if $\gamma \models \varphi$ holds for all traces $\gamma \in Traces(\mathcal{M})$.

For ease of readability, we refer to the plant strategies as *plants* and the controller strategies as *controllers*. We collect all plants in the set $\mathbb{P}$ and all controllers in the set $\mathbb{C}$. The parallel composition $\mathcal{M}_i \parallel \mathcal{M}_j$ of two strategies $\mathcal{M}_i = (S_i, s_0^i, \tau_i, o_i)$, $\mathcal{M}_j = (S_j, s_0^j, \tau_j, o_j)$ of processes $i, j \in \mathrm{Proc}$ is a strategy, i.e., a Moore machine, (S, s_0, τ, o) over inputs $(I_i \cup I_j) \setminus (O_i \cup O_j)$ and outputs $O_i \cup O_j$ with $S = S_i \times S_j$, $s_0 = (s_0^i, s_0^j)$, $\tau((s, s'), \sigma) = (\tau_i(s, (\sigma \cup o_j(s')) \cap I_i), \tau_j(s', (\sigma \cup o_i(s)) \cap I_j))$, and $o((s, s')) = o_i(s) \cup o_j(s')$.

We also represent a strategy $\mathcal{M}$ over inputs I and outputs O as a 2^O-labeled 2^I-tree which is a function $\mathcal{T} : (2^I)^* \to 2^O$. The tree maps input sequences to their last output, i.e., for all input sequences $\gamma \in (2^I)^*$, $\mathcal{T}(\gamma) = out(\mathcal{M}, \gamma)$.

ω-Automata. A (deterministic) *ω-automaton* $\mathcal{A}$ (over alphabet Σ) is a tuple (Q, q_0, δ, Ω) consisting of a finite set of states Q, an initial state $q_0 \in Q$, a transition function $\delta : Q \times \Sigma \to Q$, and an acceptance condition $\Omega \subseteq Q^\omega$. The unique *run* of $\mathcal{A}$ from state q on some trace $\gamma \in \Sigma^\infty$, denoted by $run(\mathcal{A}, q, \gamma)$, is a sequence of states $\rho \in Q^\infty$ with $|\rho| = |\gamma| + 1$, $\rho[0] = q$, and $\delta(\rho[i], \gamma[i]) = \rho[i+1]$

for all $i \in dom(\gamma)$. A run ρ is *accepting* if $\rho \in \Omega$. The language $\mathcal{L}(\mathcal{A})$ is the set of all traces γ for which the unique run $run(\mathcal{A}, q_0, \gamma)$ is accepting.

A (deterministic) safety automaton (DSA) is an ω-automaton with a safety acceptance condition $\Omega = safe(F)$, where $F \subseteq Q$ is a set of safe states, containing the set of runs that only visits states in F. It is known that an LTL specification φ can be translated into an equivalent ω-automaton $\mathcal{A}$, i.e., with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$, with a double exponential blow-up [6]. A safety LTL specification φ is an LTL specification for which the corresponding automaton $\mathcal{A}$ is a safety automaton. We write LTLtoDSA to refer to the procedure that implements this translation from a safety LTL specification to an equivalent safety automaton [34,33].

Given an automaton $\mathcal{A} = (Q, q_0, \delta, \Omega)$ and a Moore machine $\mathcal{M}$ over inputs I and outputs O with $2^{I \cup O} \subseteq \Sigma$, we define the composition $\mathcal{A} \times \mathcal{M} = (Q \times S, (q_0, s_0), \delta')$ as a product automaton with partial transition function $\delta' : (Q \times S) \times 2^I \to (Q \times S)$ defined as $\delta'((q, s), \sigma) = (\delta(q, \sigma \cup o(s)), \tau(s, \sigma))$. We write $Runs(\mathcal{A}, q, \mathcal{M})$ to denote the set of runs ρ of $\mathcal{A}$ starting from q for which there exists a corresponding run ρ' of $\mathcal{A} \times \mathcal{M}$ with $\rho'[i] = (\rho[i], s_i)$ for each $i \geq 0$. We write $\texttt{reach}(\mathcal{A} \times \mathcal{M})$ to denote the set of states (q, s) that are reachable from the initial state (q_0, s_0) in $\mathcal{A} \times \mathcal{M}$, i.e., there exists a run ρ of $\mathcal{A} \times \mathcal{M}$ with $\rho[0] = (q_0, s_0)$ and $\rho[k] = (q, s)$ for some $k \geq 0$.

Tree Automata. A tree automaton over 2^O-labeled 2^I-trees is a tuple $\mathcal{A} = (Q, q_0, T, \Omega)$ consisting of a finite set of states Q, an initial state $q_0 \in Q$, a transition function $T \subseteq Q \times 2^O \times (2^I \to Q)$, and an acceptance condition $\Omega \subseteq Q^\omega$.

A run of a tree automaton $\mathcal{A}$ on a tree $\mathcal{T} : (2^I)^* \to 2^O$ is a Q-labeled 2^I-tree $\eta : (2^I)^* \to Q$ with $\eta(\epsilon) = q_0$ and for every input sequence $\gamma\alpha \in (2^I)^*$, there exists $(\eta(\gamma), \mathcal{T}(\gamma), f) \in T$ such that $f(\alpha) = \eta(\gamma\alpha)$. Given an input sequence $\alpha_0\alpha_1 \cdots \in (2^I)^*$, a run (tree) $\eta : (2^I)^* \to Q$ produces a path, i.e., a sequence of states, $\eta(\epsilon)\eta(\alpha_0)\eta(\alpha_0\alpha_1) \cdots$. The run (tree) is accepting if every path produced by the run is accepting by the acceptance condition Ω. We define the language of a tree automaton $\mathcal{L}(\mathcal{A})$ as the set of all accepting trees.

3 Universal Controllers

Logical controller synthesis is the problem of, given a plant and an environment, finding a controller that satisfies, together with the plant, some specification. More specifically, for a given architecture, an LTL specification φ over AP, and a plant strategy $\mathcal{M}_\texttt{p}$, the problem asks to find a controller strategy $\mathcal{M}_\texttt{c}$ s.t. $\mathcal{M}_\texttt{c} \parallel \mathcal{M}_\texttt{p} \vDash \varphi$. We formalize the example from Ex. 1 as follows:

Example 2. Consider the architecture over $\texttt{AP} = \{o_\texttt{c}, o_\texttt{p}, o_\texttt{e}\}$ with $O_\texttt{e} = \{o_\texttt{e}\}$, $O_\texttt{c} = \{o_\texttt{c}\}$, $O_\texttt{p} = \{o_\texttt{p}\}$, and LTL specification φ in Eq. (1). As discussed in Ex. 1, the specification requires that the controller needs to match the output of the plant as long as the environment outputs $o_\texttt{e}$. This is also illustrated by the automaton $\mathcal{A}_\varphi$ depicted in Fig. 1 (left) which accepts $\mathcal{L}(\varphi)$ over the alphabet $\Sigma = 2^{\texttt{AP}}$. We want to synthesize a controller strategy $\mathcal{M}_\texttt{c}$ that is correct w.r.t. a given plant strategy $\mathcal{M}_\texttt{p}$, i.e., the specification is realizable by the composition of the plant and the controller.
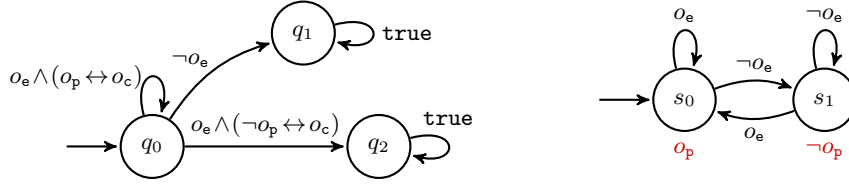
Fig. 1: Left: the (safety) automaton $\mathcal{A}_\varphi = \mathsf{LTLtoDSA}(\varphi)$ with safe states $F = \{q_0, q_1\}$ representing the specification φ in (1). Right: a plant strategy $\mathcal{M}_\mathsf{p}$.

In general, infinitely many plants exist for which one can find a controller such that the specification over the overall architecture becomes realizable. We call such plants *admissible*.

Definition 1 (Admissibility). *Given a specification φ, a plant $\mathcal{M}_p$ is said to be* admissible *for φ if there exists a controller $\mathcal{M}_c$ s.t. $\mathcal{M}_c \parallel \mathcal{M}_p \vDash \varphi$.*

Given a plant strategy $\mathcal{M}_\mathsf{p}$ and a specification φ, the *controller synthesis problem* asks to construct a controller strategy $\mathcal{M}_\mathsf{c}$ s.t. $\mathcal{M}_\mathsf{c} \parallel \mathcal{M}_\mathsf{p} \vDash \varphi$, or determine that no such controller exists, i.e., to conclude that the synthesis problem is *unrealizable*. The latter is true iff the plant $\mathcal{M}_\mathsf{p}$ is not admissible. Given a plant strategy $\mathcal{M}_\mathsf{p}$, typical controller synthesis techniques explore the entire state space of $\mathcal{M}_\mathsf{p}$ to either construct a controller or conclude unrealizability. In practice, such plant representations can be very large, as they might represent abstractions of actual implementations or physical system dynamics with a large number of states and propositions. This quickly leads to a large and infeasible runtime of synthesis algorithms.

The major observation of this work is that this exploration can be circumvented by solving the controller synthesis problem "abstractly" without considering a particular plant strategy. We achieve this by generating future assumptions about the plant. We call such future assumptions *prophecies*.

Definition 2 (Prophecies). *A prophecy $\theta \subseteq \mathbb{P}$ is a set of plants.*

Intuitively, we use prophecies to condition the current controller output on the future behavior of the plant strategy. To illustrate this, consider again the controller synthesis problem from Ex. 1. As future behavior of the environment is unknown, for any admissible plant, the controller needs to output o_c in the first time-step if the plant also outputs o_p in the first time-step, else it needs to output $\neg o_\mathsf{c}$. However, the controller cannot observe the plant output before it outputs o_c or $\neg o_\mathsf{c}$. Hence, the controller needs to use a prophecy about the plant to decide on its output. Note that the plant strategy can be very complex, involving a lot of other behavior and propositions. By synthesizing a controller which conditions its output on a *prophecy*, we extract the information about the future of the plant relevant for controller synthesis for a given specification and architecture, independently of the concrete plant realization. When given
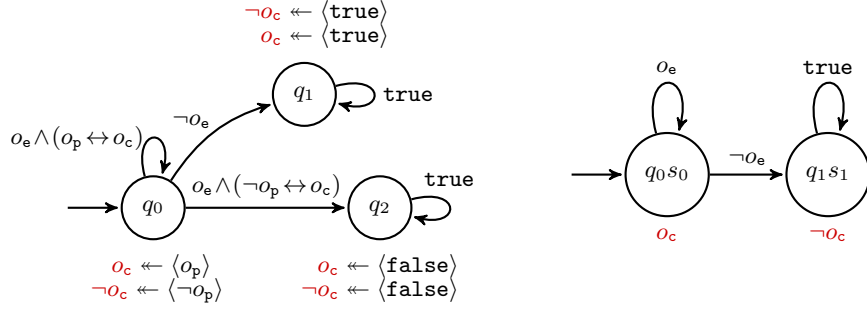
Fig. 2: Left: a universal controller $\mathcal{U}$ with prophecy annotation κ (shown by $\leftarrow$) such that $\kappa(q_1, *) = \langle \mathtt{true} \rangle$ (i.e., the set of all plants), $\kappa(q_2, *) = \langle \mathtt{false} \rangle$ (i.e., the empty set), $\kappa(q_0, o_\mathsf{c}) = \langle o_\mathsf{p} \rangle$ (i.e., the set of all plants that output o_p in the current time-step), and $\kappa(q_0, \neg o_\mathsf{c}) = \langle \neg o_\mathsf{p} \rangle$ (i.e., the set of all plants that output $\neg o_\mathsf{p}$ in the current time-step). Right: a controller strategy $\mathcal{M}_\mathsf{c}$ consistent with this universal controller w.r.t. the plant shown in Fig. 1 (right).

a concrete plant strategy, we then verify all prophecies to pick the correct controller output in every time step. We call such a prophecy-annotated controller a *universal controller*, as formalized next.

Definition 3 (Universal controller). *A* universal controller $\mathcal{U}$ *over alphabet Σ and prophecy set $\mathbb{F}$ is a tuple (S, s_0, τ, κ) consisting of a finite set of states S, an initial state $s_0 \in S$, a transition function $\tau : S \times \Sigma \to S$, and a prophecy annotation $\kappa : S \times 2^{O_\mathsf{c}} \to \mathbb{F}$.*

Note that the transition function of a universal controller is over the whole alphabet Σ, instead of just inputs of the controller. Furthermore, each state is annotated with an additional *condition* by a prophecy for each controller output. Once we obtain a plant, we can verify these prophecies to pick the correct controller output for each state and set the labelling function of the controller accordingly. Once the labelling function is set, this also refines the transition function into a function over the inputs only.

Example 3. Consider again the controller synthesis problem from Ex. 1 and a universal controller $\mathcal{U}$ as shown in Fig. 2 (left). In the first time-step, as future behavior of the environment is unknown, the controller must set o_c to true if the plant also sets o_p to true. Hence, the prophecy for the output o_c at the initial state q_0 in Fig. 2 (left), i.e., $\kappa(q_0, o_\mathsf{c}) = \langle o_\mathsf{p} \rangle$, is the set of all plants that output o_p in the current time-step. Similarly, the prophecy for the output $\neg o_\mathsf{c}$ at state q_0, i.e., $\kappa(q_0, \neg o_\mathsf{c}) = \langle \neg o_\mathsf{p} \rangle$, is the set of all plants that output $\neg o_\mathsf{p}$ in the current time-step. Furthermore, once the environment outputs $\neg o_\mathsf{e}$, we reach the state q_1 where the controller can set any output as the specification is satisfied no matter what the controller does. The prophecy for both outputs o_c and $\neg o_\mathsf{c}$ at state q_1 is $\langle \mathtt{true} \rangle$, i.e., it can set either output for any plant. Similarly, if the

environment outputs o_{e}, and the output of the plant and the controller does not match, we reach the state q_2 where the specification is violated no matter what the controller does. Hence, the prophecy for both output o_{c} and $\neg o_{\mathsf{c}}$ at state q_2 is $\langle \mathtt{false} \rangle$, i.e., there is no output that is correct for any plant.

Such a universal controller is *universal* in the sense that it represents a set of controllers for each plant as formalized next.

Definition 4 (Consistency). *Given a universal controller $\mathcal{U} = (S, s_0, \tau, \kappa)$ and a plant $\mathcal{M}_p$, a controller $\mathcal{M}_c = (S^c, s_0^c, \tau^c, o^c)$ is said to be* consistent *with $\mathcal{U}$ w.r.t. $\mathcal{M}_p$, denoted by $\mathcal{M}_c \vDash \mathcal{U} \parallel \mathcal{M}_p$, if for all $(s, s^p, s^c) \in \mathit{reach}(\mathcal{U} \times (\mathcal{M}_p \parallel \mathcal{M}_c))$, it holds that $\mathcal{M}_p(s^p) \in \kappa(s, o^c(s^c))$.*

Intuitively, at each state of the universal controller, the corresponding plant state satisfies the prophecy for the controller output from that state. Given a universal controller $\mathcal{U}$ and a plant $\mathcal{M}_p$, one can find a controller $\mathcal{M}_c$ such that $\mathcal{M}_c \vDash \mathcal{U} \parallel \mathcal{M}_p$ (if one exists) by verifying the plant against each prophecy in the current state of the universal controller and picking a controller output for which the corresponding prophecy is satisfied by the plant.

Example 4. Consider the universal controller $\mathcal{U}$ shown in Fig. 2 (left) and the plant strategy $\mathcal{M}_p$ shown in Fig. 1 (right). We can construct a controller strategy $\mathcal{M}_c$ that is consistent with $\mathcal{U}$ w.r.t. $\mathcal{M}_p$ by verifying the prophecies as follows. From the initial state q_0, the plant $\mathcal{M}_p$ sets o_{p} to true and hence, satisfies the prophecy $\kappa(q_0, o_{\mathsf{c}})$. Thus, a consistent controller has to set o_{c} to true in the first time-step. Later on from state q_0, the plant $\mathcal{M}_p$ continues setting o_{p} to true as long as the environment outputs o_{e}. Hence, a consistent controller also has to set o_{c} to true as long as the environment outputs o_{e}. As soon as the environment outputs $\neg o_{\mathsf{e}}$, we reach state q_1 in the universal controller $\mathcal{U}$ from which the prophecy for o_{c} and $\neg o_{\mathsf{c}}$ are $\langle \mathtt{true} \rangle$, and hence, the controller can set any output. By picking the prophecy $\kappa(q_1, \neg o_{\mathsf{c}})$, the controller $\mathcal{M}_c$ sets o_{c} to false for all future time-steps. With this, we obtain the controller strategy $\mathcal{M}_c$ shown in Fig. 2 (right) which is consistent with $\mathcal{U}$ w.r.t. $\mathcal{M}_p$. Note that as we set controller outputs by verifying the prophecies for the plant $\mathcal{M}_p$, we never had to verify prophecies at the state q_2 in the universal controller $\mathcal{U}$ where the specification is violated.

Given the above representation of controllers by universal controllers, our main goal is to find a universal controller such that given any admissible plant, the controller strategy consistent with them solves the logical controller synthesis problem. We call such a universal controller *correct* as formalized below.

Definition 5 (Correctness). *Given a specification φ, a universal controller $\mathcal{U}$ is* correct *for φ if for all admissible plant strategies $\mathcal{M}_p$, it holds that $\mathcal{M}_c \vDash \mathcal{U} \parallel \mathcal{M}_p$ implies $\mathcal{M}_p \parallel \mathcal{M}_c \vDash \varphi$.*

Conversely, if all correct controllers for an admissible plant are consistent with the universal controller, we call such a universal controller *most permissive*.

Definition 6 (Permissiveness).   *Given a specification φ, a universal controller $\mathcal{U}$ is most permissive for φ if for all admissible plant strategies $\mathcal{M}_p$, it holds that $\mathcal{M}_p \parallel \mathcal{M}_c \vDash \varphi$ implies $\mathcal{M}_c \vDash \mathcal{U} \parallel \mathcal{M}_p$.*

Example 5. Consider the universal controller $\mathcal{U}$ shown in Fig. 2 (left), the plant strategy $\mathcal{M}_p$ shown in Fig. 1 (right), and the controller strategy $\mathcal{M}_c$ shown in Fig. 2 (right) that is consistent with $\mathcal{U}$ w.r.t. $\mathcal{M}_p$. The universal controller $\mathcal{U}$ is correct and most permissive for the specification φ in (1) and hence, the controller strategy $\mathcal{M}_c$ solves the logical controller synthesis problem for φ with given plant strategy $\mathcal{M}_p$.

We define the formal problem statement as follows:

Definition 7 (Universal controller synthesis). *Given a specification φ, the problem of* universal controller synthesis *is to find a universal controller that is correct and most permissive for φ.*

Computing a universal controller reduces to the problem of computing suitable prophecies. The following section presents an automata-based algorithm to compute prophecies for safety specifications. These prophecies build the foundation of our algorithm for synthesizing universal controllers.

4   Prophecies for Controller Synthesis

In this section, we introduce a class of prophecies, namely *safe prophecies*, which are sufficient to construct a universal controller for safety specifications that is both correct and most permissive.

4.1   Prophecies

The class of safe prophecies only considers the current and next state of execution to capture the safety part of the given specification. Intuitively, for the current choice of the controller output, a safe prophecy collects a set of plants for which this choice is *safe*, i.e., it can lead to an accepting run in the automaton. We define these prophecies for a given automaton:

Definition 8. *Let $\mathcal{A} = (Q, q_0, \delta, \Omega)$ be an automaton, $q \in Q$ be a state in $\mathcal{A}$, and $\alpha \in 2^{O_c}$ be an output of the controller. The* safe prophecy $\mathbb{S}_{\mathcal{A}}^{q,\alpha}$ *is the set*

$$\mathbb{S}_{\mathcal{A}}^{q,\alpha} = \{\mathcal{M}_p \in \mathbb{P} \mid \exists \mathcal{M}_c \in \mathbb{C}.\ out(\mathcal{M}_c, \epsilon) = \alpha \wedge Runs(\mathcal{A}, q, \mathcal{M}_p \parallel \mathcal{M}_c) \subseteq \Omega\}.$$

Given a state $q \in Q$ and a controller output $\alpha \in 2^{O_c}$, the set $\mathbb{S}_{\mathcal{A}}^{q,\alpha}$ collects all plant strategies for which there exists a controller strategy $\mathcal{M}_c$ from q that is consistent with the current output α and produces only accepting runs with $\mathcal{M}_p$ in the automaton, i.e., $Runs(\mathcal{A}, q, \mathcal{M}_p \parallel \mathcal{M}_c) \subseteq \Omega$. Intuitively, the safe prophecy of a controller's choice collects all the plant strategies for which this choice is safe. Hence, for any safety LTL formula, safe prophecies provide necessary restrictions on the controller's choices.

Theorem 1. *Let φ be a safety LTL specification and $\mathcal{A} = (Q, q_0, \delta, \Omega = safe(F))$ its equivalent safety automaton. The universal controller $\mathcal{U} = (Q, q_0, \delta, \kappa)$ with $\kappa(q, \alpha) = \mathbb{S}_{\mathcal{A}}^{q, \alpha}$ is most permissive for φ.*

Proof. Let $\mathcal{M}_{\mathsf{p}} = (S^{\mathsf{p}}, s_0^{\mathsf{p}}, \tau^{\mathsf{p}}, o^{\mathsf{p}})$ be an admissible plant and let the controller strategy $\mathcal{M}_{\mathsf{c}} = (S^{\mathsf{c}}, s_0^{\mathsf{c}}, \tau^{\mathsf{c}}, o^{\mathsf{c}})$ be such that $\mathcal{M}_{\mathsf{p}} \parallel \mathcal{M}_{\mathsf{c}} \vDash \varphi$. We need to show that $\mathcal{M}_{\mathsf{c}} \vDash \mathcal{U} \parallel \mathcal{M}_{\mathsf{p}}$. Consider the composition $\mathcal{U}' = \mathcal{U} \times (\mathcal{M}_{\mathsf{p}} \parallel \mathcal{M}_{\mathsf{c}}) = (S', s_0', \tau')$. Let $s' = (q, s^{\mathsf{p}}, s^{\mathsf{c}}) \in \mathtt{reach}(\mathcal{U}')$. Then, it is enough to show that $\mathcal{M}_{\mathsf{p}}(s^{\mathsf{p}}) \in \kappa(q, o^{\mathsf{c}}(s^{\mathsf{c}})) = \mathbb{S}_{\mathcal{A}}^{q, o^{\mathsf{c}}(s^{\mathsf{c}})}$. This means, it is enough to show that for strategy $\mathcal{M}_{\mathsf{c}}' = \mathcal{M}_{\mathsf{c}}(s^{\mathsf{c}})$ and $\mathcal{M}_{\mathsf{p}}' = \mathcal{M}_{\mathsf{p}}(s^{\mathsf{p}})$, it holds $Runs(\mathcal{A}, q, \mathcal{M}_{\mathsf{p}}' \parallel \mathcal{M}_{\mathsf{c}}') \subseteq \Omega$.

Let ρ be a run in $Runs(\mathcal{A}, q, \mathcal{M}_{\mathsf{p}}' \parallel \mathcal{M}_{\mathsf{c}}')$, then by definition, there exists a run ρ' in $\mathcal{U}'$ with $\rho'[i] = (\rho[i], *, *)$ for all $i \geq 0$. Let $\rho^f = (q_0, s_0^{\mathsf{p}}, s_0^{\mathsf{c}}) \dots (q_k, s_k^{\mathsf{p}}, s_k^{\mathsf{c}})$ be a path in $\mathcal{U}'$ from s_0' to s'. Then, by construction, $(\rho^f[0, k-1])\rho'$ is a run from s_0' in $\mathcal{U}'$. As $\mathcal{M}_{\mathsf{c}} \parallel \mathcal{M}_{\mathsf{p}} \vDash \varphi$ and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$, the corresponding run of $(\rho^f[0, k-1])\rho'$ in $\mathcal{A}$ is accepting, i.e., $q_0 q_1 \dots q_{k-1}\rho \in \Omega$. By the definition of safety acceptance, we have $\rho \in \Omega$. As ρ is arbitrary, we have $Runs(\mathcal{A}, q, \mathcal{M}_{\mathsf{p}}' \parallel \mathcal{M}_{\mathsf{c}}') \subseteq \Omega$, and hence, $\mathcal{M}_{\mathsf{c}} \vDash \mathcal{U} \parallel \mathcal{M}_{\mathsf{p}}$. $\qquad\square$

The theorem shows that every controller strategy that is correct for a plant is preserved in the prophecy set. Furthermore, as ensuring that the controller does not make any unsafe choices is sufficient to realize a safety specification, we next show that the safe prophecies are sufficient to synthesize a correct universal controller realizing any safety specification once a plant strategy is given.

Theorem 2. *Let φ be a safety LTL specification and $\mathcal{A} = (Q, q_0, \delta, \Omega = safe(F))$ its equivalent safety automaton. The universal controller $\mathcal{U} = (Q, q_0, \delta, \kappa)$ with $\kappa(q, \alpha) = \mathbb{S}_{\mathcal{A}}^{q, \alpha}$ is correct for φ.*

Proof. Let $\mathcal{M}_{\mathsf{p}} = (S^{\mathsf{p}}, s_0^{\mathsf{p}}, \tau^{\mathsf{p}}, o^{\mathsf{p}})$ be an admissible plant and let the controller strategy $\mathcal{M}_{\mathsf{c}} = (S^{\mathsf{c}}, s_0^{\mathsf{c}}, \tau^{\mathsf{c}}, o^{\mathsf{c}})$ be consistent with $\mathcal{U}$ w.r.t. $\mathcal{M}_{\mathsf{p}}$, i.e., $\mathcal{M}_{\mathsf{c}} \vDash \mathcal{U} \parallel \mathcal{M}_{\mathsf{p}}$. We need to show that $\mathcal{M}_{\mathsf{p}} \parallel \mathcal{M}_{\mathsf{c}} \vDash \varphi$. Let $\gamma \in Traces(\mathcal{M}_{\mathsf{p}} \parallel \mathcal{M}_{\mathsf{c}})$ with $\rho = run(\mathcal{A}, q_0, \gamma)$ being the corresponding unique run. As $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$, to prove $\gamma \in \mathcal{L}(\varphi)$, we only need to show that $\rho \in \Omega$, i.e., $\rho[i] \in F$ for all $i \geq 0$.

Let $\mathcal{M}_{\mathsf{p}} = (S^{\mathsf{p}}, s_0^{\mathsf{p}}, \tau^{\mathsf{p}}, o^{\mathsf{p}})$ and $\mathcal{M}_{\mathsf{c}} = (S^{\mathsf{c}}, s_0^{\mathsf{c}}, \tau^{\mathsf{c}}, o^{\mathsf{c}})$. Then, as $\mathcal{M}_{\mathsf{c}} \vDash \mathcal{U} \parallel \mathcal{M}_{\mathsf{p}}$, by definition, for every $(q, s^{\mathsf{p}}, s^{\mathsf{c}}) \in \mathtt{reach}(\mathcal{U} \times (\mathcal{M}_{\mathsf{p}} \parallel \mathcal{M}_{\mathsf{c}}))$, it holds that $\mathcal{M}_{\mathsf{p}}(s^{\mathsf{p}}) \in \kappa(q, o^{\mathsf{c}}(s^{\mathsf{c}})) = \mathbb{S}_{\mathcal{A}}^{q, o^{\mathsf{c}}(s^{\mathsf{c}})}$.

As $\gamma \in Traces(\mathcal{M}_{\mathsf{p}} \parallel \mathcal{M}_{\mathsf{c}})$, by construction, there exists a run ρ' from the initial state in $\mathcal{U} \times (\mathcal{M}_{\mathsf{p}} \parallel \mathcal{M}_{\mathsf{c}})$ corresponding to γ such that $\rho'[i] = (\rho[i], s_i^{\mathsf{p}}, s_i^{\mathsf{c}})$. Hence, for all $i \geq 0$, we have $\rho'[i] \in \mathtt{reach}(\mathcal{U} \times (\mathcal{M}_{\mathsf{p}} \parallel \mathcal{M}_{\mathsf{c}}))$, and hence, $\mathcal{M}_{\mathsf{p}}(s_i^{\mathsf{p}}) \in \mathbb{S}_{\mathcal{A}}^{\rho[i], o^{\mathsf{c}}(s_i^{\mathsf{c}})}$. Then, by definition of safe prophecies, for each $i \geq 0$, there exists $\mathcal{M}_{\mathsf{c}}^i \in \mathbb{C}$ with $out(\mathcal{M}_{\mathsf{c}}^i, \epsilon) = o^{\mathsf{c}}(s_i^{\mathsf{c}})$ and $Runs(\mathcal{A}, \rho[i], \mathcal{M}_{\mathsf{c}}^i \parallel \mathcal{M}_{\mathsf{p}}(s_i^{\mathsf{p}})) \subseteq \Omega$. This means, for each $i \geq 0$, there exists runs from $\rho[i]$ in $\mathcal{A}$ that are accepting, and hence, by definition of safety acceptance Ω, we have $\rho[i] \in F$. $\qquad\square$

The universal controller for our running example shown in Fig. 2 (left) is annotated with safe prophecies and hence, is correct and most permissive for the safety specification φ in (1).

4.2   Automata for Prophecies

Prophecies, as defined in Def. 2 and Def. 8, define sets of plants that are correct for an explicit controller choice. A suitable automaton model for prophecies are tree automata whose languages are sets of trees (in contrast to sets of traces for word automata). Every tree in the language of such an automaton corresponds to a strategy of a plant, which is also an infinite tree. We show that we can construct a tree automaton that correctly accepts the safe prophecy for a given controller choice. The basic idea of the construction is similar to the automaton to two-player game conversion for LTL synthesis [20]. A plant satisfies the prophecy if it has a correct reaction to any choice of the controller inputs. Additionally, we restrict the output of the controller in the first step of the automaton, i.e., the output we compute the prophecy for. This yields the following result.

Theorem 3. *Let $\mathcal{A} = (Q, q_0, \delta, \Omega)$ be a safety automaton for the safety LTL formula φ and $q_s \in Q, \alpha_s \in 2^{O_c}$ be an output of the controller. There exists a linear tree automaton $\mathcal{A}'$ s.t. $\mathcal{L}(\mathcal{A}') = \mathbb{S}_{\mathcal{A}}^{q_s, \alpha_s}$.*

Proof. We construct a tree automaton $\mathcal{A}'$ over (2^{O_p})-labelled 2^{I_p} trees that accepts all plants for which a controller strategy exists with the current choice α_s that ensures the specification. The idea is to simulate the plant and controller behavior on the word automaton $\mathcal{A}$, where the automaton is implicitly labelled with both plant and controller outputs, and only branches over environment outputs. To make the tree automaton labelled with plant outputs, we remove each controller output β_c from the labels and make the corresponding node branch over all possible plant inputs, i.e., outputs of both controller and environment, by taking all branches with inconsistent controller outputs (i.e., $\neq \beta_c$) to an accepting sink state. In addition, we simulate $\mathcal{A}$ from q_s and restrict the initial choice of the controller to α_s by introducing a new initial state q_0'. Formally, the automaton $\mathcal{A}' = (Q', q_0', T', \Omega')$ is defined as follows:

- $Q' = Q \uplus \{q_0', q_{sink}\}$,
- $T' \subseteq Q \times 2^{O_p} \times (2^{I_p} \to Q)$ is the minimal set such that
  - for all $\beta_p \in 2^{O_p}$, there is a transition (q_0', β_p, f) such that for all $\beta \in 2^{I_p}$,
  $$f(\beta) = \begin{cases} \delta(q_s, \beta_p \cup \beta) & \text{if } \beta \cap O_c = \alpha_s, \\ q_{sink} & \text{otherwise,} \end{cases}$$
  - for all $q \in Q$, $\beta_p \in 2^{O_p}$, and $\beta_c \in 2^{O_c}$, there is a transition (q, β_p, f) such that for all $\beta \in 2^{I_p}$, $f(\beta) = \begin{cases} \delta(q, \beta_p \cup \beta) & \text{if } \beta \cap O_c = \beta_c, \\ q_{sink} & \text{otherwise,} \end{cases}$
  - for all $\beta_p \in 2^{O_p}$, there is a transition (q_{sink}, β_p, f) such that $f(\beta) = q_{sink}$ for all $\beta \in 2^{I_p}$.
- $\Omega' = safe(F')$ with $F' = F \cup \{q_0', q_{sink}\}$.

Compared to $\mathcal{A}$, the automaton $\mathcal{A}'$ has two new states q_0' and q_{sink}. For all transitions from q_0', we fix the controller output to be α_s and quantify over all possible plant outputs. Afterwards, we branch over all possible plant inputs but

only simulate the automaton $\mathcal{A}$ when the controller output is α_s, otherwise, we go to the sink state q_{sink}. Similarly, in the general case, we quantify over all possible controller outputs β_c and plant outputs β_p, but only use the plant output to explicitly label the tree. Then, we branch over all possible plant inputs but only simulate the automaton $\mathcal{A}$ when the controller output is β_c, otherwise, we go to the sink state q_{sink}. The runs of this automaton are all trees that branch over environment and controller outputs and are labeled with plant outputs that satisfy the prophecy $\mathbb{S}_{\mathcal{A}}^{q_s,\alpha_s}$. Since the transition function follows a similar construction for reactive synthesis (without a plant) in [20], the correctness of the construction follows immediately. As $|Q'| = |Q| + 2$, the size of $\mathcal{A}'$ is linear in the size of $\mathcal{A}$. □

Note that it is not possible to represent safe prophecies with a word automaton as we are looking for an automaton that accepts all plant strategies of any size, therefore infinite trees. One key insight is that we do not only have to verify the choices of the controller against the behavior of the plant, but, there must also exist a strategy of the controller such that all future behavior of the plant sat-

Fig. 3: A part of the tree automaton (without transitions of q_0 and q_{sink}) with safe states $F' = \{q_0', q_0, q_1\}$ for prophecy $\mathbb{S}_{\mathcal{A}_\varphi}^{q_0,o_c} = \langle o_p \rangle$ of Fig. 2, where each function f_i shown as dashed edges.

isfies the specification. Both of these criteria are captured in the constructed tree automaton.

Example 6. Consider the universal controller in Fig. 2 (left) for the running example. A tree automaton $\mathcal{A}'$ for prophecy $\kappa(q_0, o_c) = \mathbb{S}_{\mathcal{A}_\varphi}^{q_0,o_c}$ is shown in Fig. 3. As we are considering a prophecy from the initial state q_0, we have a new initial state q_0' that is a copy of q_0. From q_0', we fix the controller output to be o_c for simulation, and branch over all possible plant outputs. Given plant output o_p from q_0', we simulate the automaton $\mathcal{A}_\varphi$ by function f_1 when the controller output is o_c (else we go to the sink state q_{sink}). Hence, it branches to state q_0 and q_1 depending on the environment outputs. Similarly, given plant output $\neg o_p$ from q_0', we simulate the automaton $\mathcal{A}_\varphi$ by function f_2 when the controller output is o_c. From every other state, we branch over all possible plant and controller outputs. For example, from state q_1, we fix the controller output to be o_c (resp. $\neg o_c$) and simulate the automaton $\mathcal{A}_\varphi$ by function f_3 (resp. f_4) with consistent controller output. As every transition in $\mathcal{A}_\varphi$ from q_1 leads to q_1, the function f_3 maps every input with consistent controller output to q_1 (and inconsistent controller output to q_{sink}).

Once we have the tree automaton for the prophecy, we can use it to construct an explicit controller consistent with a universal controller by verifying the plant

strategies against the tree automaton. For verifying a plant against such a tree automaton, we check if the infinite tree produced by the plant (strategy) is an element of the language of the tree automaton. We refer to the procedure that constructs the automaton for the prophecy as $\texttt{prophecy}(q, \alpha, \mathcal{A})$.

5   Universal Controller Synthesis

In this section, we present an automata-theoretic approach to synthesizing correct universal controllers from safety LTL formulas. We also present an algorithm that, given a plant and a universal controller, constructs a controller that is consistent with the plant and satisfies the specification.

5.1   Universal Controllers

Algo. 1 shows the main procedure for the construction of a universal controller. The first step is to construct an equivalent safety automaton $\mathcal{A}_\varphi$ for the specification φ. The state space of the automaton yields the initial state space for the

---
Algorithm 1 Universal Controller Synthesis

```
1  let universalController(φ) :=
2     let 𝒜_φ = (Q, q₀, δ, Ω) ← LTLtoDSA(φ)
3     let κ ← []
4     for q ∈ Q and α ∈ 2^{O_c}
5        let κ(q, α) ← prophecy(q, α, 𝒜_φ)
6     return 𝒰 = (Q, q₀, δ, κ)
```

---

universal controller. The algorithm continues with iterating over every pair of state and controller output, and builds the respective prophecy with the construction introduced in Section 4.2. Once the map κ contains all pairs of states and outputs, we build the universal controller $\mathcal{U}$. The universal controller can be interpreted as follows: For a given environment output and state q, some output α is taken for which the plant behaves as specified in the tree automaton $\kappa(q, \alpha)$. Since the alphabet of the automaton $\mathcal{A}_\varphi$ ranges over all propositions, the transition function δ is also defined over the plant outputs. During composition, we ensure that the prophecy also agrees with the current output of the plant to ensure consistency.

Subautomata Sharing. Instead of constructing a new tree automaton for every state-output pair in $\mathcal{A}_\varphi$, we build a global tree automaton that contains all possible plants for the specification and only set new initial states and transitions. The global tree automaton construction is similar to $\texttt{prophecy}(q, \alpha, \mathcal{A})$ but q is the initial state q_0 of $\mathcal{A}_\varphi$ and α is not considered in the construction. One obtains the tree automaton $\Omega_\mathcal{T} = (Q, q_0, T, \Sigma)$ that accepts all plant and controller pairs that are accepting for the specification. In the loop of Algo. 1, the only construction step is changing the initial state in $\mathcal{A}_\mathcal{T}$ to a duplicate of q and adding an initial step from q with α to the transition relation T. For the implementation of this algorithm, as presented in Section 6, it suffices to store the new initial state and the controller outputs. The modified copy of the global tree automaton is only computed during the verification in Algo. 2.

5.2   Composition

Algorithm 2 On-the-fly Composition

Once the universal controller is computed, we can compose it with a plant to obtain a plant-specific controller that is consistent with the plant's behavior. The algorithm operates on an on-the-fly exploration of the product state-space of the universal controller and the

```
1  let composition(𝒰 = (Q, q₀, δ, κ), ℳₚ = (Q', q'₀, δ', o)) :=
2     let s ← (q₀, q'₀), states ← Set(), edges ← Set()
3     let queue ← Queue(s)
4     while (q_c, q_p) ← queue.pop() do
5        let edgelist ← stepComposition(q_c, q_p, 𝒰, ℳₚ)
6        for (σ, (q'_c, q'_p)) in edgelist do
7           𝒜_𝒯 ← κ(q_c, σ)
8           ℳ'ₚ ← modifiedPlant(ℳₚ, q_p, σ)
9           if isMember(ℳ'ₚ, 𝒜_𝒯) then
10             edges.add((q_c, q_p), σ, (q'_c, q'_p))
11             queue.push(q'_c, q'_p) if not states.contains(q'_c, q'_p)
12          states.add(q'_c, q'_p)
13     return (states, (q₀, q'₀), edges)
```

plant and is shown in Algo. 2. The new initial state of the controller is a tuple (q_0, q'_0), i.e., the initial states of the universal controller and the plant. Note that we omit double parentheses for function calls over tuple of universal controller and plant states whenever it is clear from the context. We use a **while** loop over the states to be explored in the composition as a termination condition. The body of the loop that ranges from line 5 to line 12 starts by computing a one-step composition from the current state (q_c, q_p) in $\texttt{stepComposition}(q_c, q_p, \mathcal{U}, \mathcal{M}_\mathtt{p})$ which takes the states, the universal controller, and the plant as input. The result of this function is a list of edges, i.e., pairs over an assignment of the alphabet and future state, in the composition that are possible to reach. However, since not every computed edge is correct for the given plant, the prophecy of this explicit edge has to be checked first before adding it to the controller's edges. To verify the plant, we first obtain the tree automaton $\mathcal{A}_\mathcal{T}$ representing the prophecy with $\kappa(q_c, \sigma)$ and change the input plant to a modified version $\mathcal{M}'_\mathtt{p}$ that starts in q_p with σ. The membership check is then performed in line 9 with a call to $\texttt{isMember}(\mathcal{M}'_\mathtt{p}, \mathcal{A}_\mathcal{T})$ returning true if the tree produced by $\mathcal{M}_\mathtt{p}$ is an accepting run of $\mathcal{A}_\mathcal{T}$. In case of the membership check being successful, the edge is a correct edge in the controller and is added to the set of edges and, if not already explored, the target of the edge is added to the queue. The result of the algorithm is then all explored states and edges with the starting state (q_0, q'_0).

Tree Automata Checking. The most involved computation is verifying the modified plant against the prophecy tree automaton. In Algo. 2, this check is performed in the function $\texttt{isMember}(\mathcal{M}'_\mathtt{p}, \mathcal{A}_\mathcal{T})$. Following the literature [30], we encode this function into a 2-player game where the environment player chooses the inputs of the plant and the system player has to provide a correct output for every choice of the environment. If the system player wins, then the infinite tree produced by the plant's Moore machine is contained in the language of the tree automaton. The winning condition of the game is *safety*, implied by the acceptance condition of the tree automaton, and the state space is linear in the size of the tree automaton and the plant.

Hashing. The number of automaton checks of Algo. 2 is $|\Sigma| \times |\mathcal{U}| \times |\mathcal{M}_\mathtt{p}|$, i.e., the product of the alphabet size, the universal controller size, and the plant size. We reduce the number of automata checks drastically by hashing intermediate results: For a solved game we obtain a set of *winning* and a set of *losing* states in the game. These states can be directly mapped back to the composition such that we maintain a function $\mathtt{h}(\sigma, (q_c, q_p))$ that stores for already computed edge-state pairs if the game is winning or losing. Then, before calling `isMember`, we check if the current edge was hashed before and, if so, return the hashed result. In our experiments in Section 6, we observe that the number of *real* containment checks that are computed without hashing is *one* for almost all benchmarks. This is the case if every state is reachable for every initial transition from (q_0, q_0') since this is the first state that we compute the safety game for.

Early Termination. The local exploration of the state space allows the use of heuristics for termination during both, composition of universal controller with plant and construction of the game. One such heuristics is defined over the transition function of currently discovered edges: Once the explored set of edges has a transition for every possible output of the environment, then we can stop exploring the current edges and move on to the next state in the queue. We do this check before jumping from line 11 to line 6. Furthermore, we order the edgelist according to already explored states so that we look for such edges before verifying edges to new states. The semantics of the specification is used to reduce the size of the game to be solved during verification. Since the acceptance condition is solely based on the specification automaton, we can stop exploring the state space of the plant once we are in a clearly accepting/rejecting loop, e.g., a terminal state. The early termination heuristics all benefit from one key observation: the specifics of the plant implementation can often be ignored while solving the controller synthesis problem.

Note that for simplicity, we construct the controller as a Mealy machine, i.e., as usual for automata, we use edge labels instead of state labels. However, this can easily be transformed into Moore machines by ignoring the first environment output and moving the controller output to the source state of every edge.

5.3   Correctness and Complexity

Algo. 1 and Algo. 2 together compute a controller for a given specification and plant with the automata constructions presented in Section 4.2. The correctness of the algorithm immediately follows from (Thm. 2). As the tree automata for the prophecies are linear in the size of the deterministic specification automaton, solving the game in polynomial time results in the same complexity as the standard controller synthesis problem [20]. Moreover, as the number of prophecies is linear in the size of the specification automaton, the overall time complexity of universal controller synthesis remains doubly exponential in the size of the specification. In summary, we observe the following result:

Corollary 1. *Let φ be a safety LTL formula. Algo. 1 returns a universal controller $\mathcal{U}$ that is correct and most permissive for φ. Furthermore, given an ad-*

(a) Specification Type 1

(b) Specification Type 2

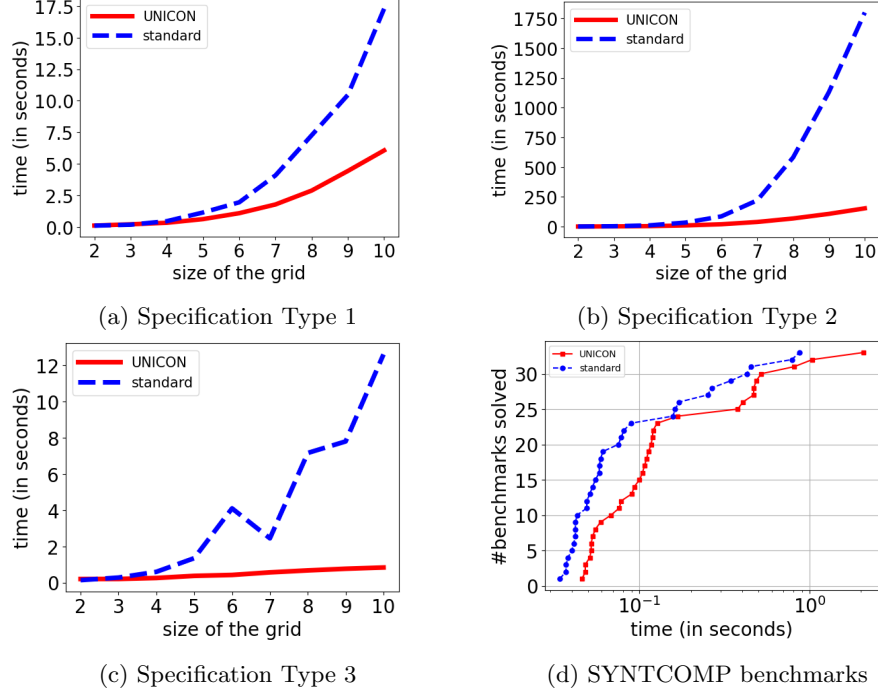
(c) Specification Type 3

(d) SYNTCOMP benchmarks

Fig. 4: We compare the execution times of UNICON and the standard synthesis approach. Fig. 4a to Fig. 4c show the results of three different specification types on increasing robot benchmarks. Fig. 4d shows a cactus plot for the number of SYNTCOMP benchmarks solved in a certain time budget.

missible plant $\mathcal{M}_p$, Algo. 2 returns a controller $\mathcal{M}_c$ such that $\mathcal{M}_c \parallel \mathcal{M}_p \vDash \varphi$. The overall time complexity of computing $\mathcal{M}_c$ using Algo. 1 and Algo. 2 is doubly-exponential in the size of the specification φ and linear in the size of the plant $\mathcal{M}_p$.

6   Experiments

We have implemented our approach in a prototype tool called UNICON [24]. For benchmarking purposes, UNICON additionally implements the standard controller synthesis algorithm that reduces the synthesis problem to game solving [20]. The tool is implemented in F# and uses FsOmegaLib [1] and SPOT [18] for automata operations and OINK [17] for game solving. The experiments are performed on a computer with an Apple M1 Pro 8-core CPU and 16GB of RAM.

We evaluate our approach on a set of benchmarks where the controller guides a robot through a maze-like workspace, avoiding collisions with the walls and with a second (uncontrollable) robot. Given the size of the grid as a parameter, the workspace is encoded in the plant model as a grid of cells with some walls

between the cells and walls around the boundary of the grid. Initially, the controller robot is placed in the center of the grid and the plant robot in the bottom left corner. At each step, the robots can either stay or move in four directions: up, down, left, and right. The controller robot has sensors to detect the obstacles (i.e., walls or the plant robot) if they are adjacent to it or when there is a collision. The behavioral objective of the controlled robot is specified as an LTL formula. We consider three different types of specifications: Specification type 1 is an invariant $\square \neg col$, which simply states that the controller must always avoid collisions. In specification type 2, we add some more information about the setting into the temporal formula: $\neg col \; W \; \neg \mathtt{assump_p}$, where the assumption $\mathtt{assump_p}$ expresses that a collision can occur only if in the previous step there was an obstacle in some direction and the controller moved in that direction or stayed in the same position (which might lead to a collision if the obstacle was the other robot). Specification type 3 is a condition that only restricts the first three steps in the execution, requiring that during those three steps, the robot does not see an obstacle in the up direction: $\bigcirc \neg u_{obs} \wedge \bigcirc\bigcirc \neg u_{obs} \wedge \bigcirc\bigcirc\bigcirc \neg u_{obs}$.

Evaluation on robot benchmarks. The results of the experiments are shown in Fig. 4, which shows the comparison of the execution times of UNICON (i.e., the total time for synthesizing the universal controller and extracting a correct controller using Algos. 1 and 2) with the standard controller synthesis approach to compute a correct controller. In all three types of specifications, our approach scales well with growing grid size, while the running time of the standard approach quickly becomes prohibitively large. The reason is that standard controller synthesis constructs the complete composition of the plant and the specification automaton to solve the synthesis problem, while the universal controller synthesis composes the plant and the prophecies on-the-fly. Since the prophecy size is independent of the plant size, the universal controller synthesis scales better with the increasing grid size. Specification type 2 performs better than specification type 1, as the added information of the assumption allows for simpler prophecies. Simple specifications like type 3 are most beneficial as the prophecies only need to consider three steps into the future.

Baseline evaluation on SYNTCOMP benchmarks. The benchmarks of the annual SYNTCOMP reactive synthesis competition [27] are often used to evaluate synthesis algorithms. Since the benchmarks do not contain plant models, we cannot use them to evaluate the scalability of our approach, but we still include them here as a baseline to evaluate the overhead caused by the computation of the universal controller. We selected the 68 benchmarks that have safety requirements and explicit environment assumptions, and translated the assumption formulas into plant models. As shown in Fig. 4d, UNICON shows similar performance to standard synthesis. Because of the small size of the plant models resulting from the assumption formulas, the overhead for the universal controller is not compensated by the improved performance in the size of the plant model. Still, computation times remain close, showing that UNICON is able to scale to practical problem instances.

7   Conclusions

Universal controllers have three major advantages over standard control strategies: The first advantage is the improved scalability due to the fact that the universal controller is constructed *independently of the plant*. There is no free lunch – adapting the universal controller to a specific plant does require an analysis of the plant model. However, we only need to verify the prophecy, not the original specification. Simple prophecies can be checked without a full exploration of the state space. There is no loss of completeness: the adaptation of the universal controller to a particular plant succeeds whenever the synthesis problem is realizable for the plant.

A second advantage is that the universal controller is *adaptable* to new plants; if the plant model changes, it is not necessary to re-synthesize the universal controller, only the prophecies need to be re-evaluated. In principle, the controller can even be adapted to dynamic changes in the plant by evaluating the prophecies on-the-fly.

A third major advantage of the universal controller is its *explainability*: the prophecy identifies the specific property of the plant that makes it safe to choose the move. If the adaptation fails, the violated prophecies identify properties of the plant that make it unrealizable. In future work, we plan to exploit these advantages by constructing controllers that adapt to new situations dynamically and explain their behavior to the user in terms of these changes.

References

1. Fsomegalib. https://github.com/ravenbeutner/FsOmegaLib, accessed: 2024-09-25
2. Abadi, M., Lamport, L.: The existence of refinement mappings. Theor. Comput. Sci. **82**(2), 253–284 (1991). https://doi.org/10.1016/0304-3975(91)90224-P, https://doi.org/10.1016/0304-3975(91)90224-P
3. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Ausiello, G., Dezani-Ciancaglini, M., Rocca, S.R.D. (eds.) Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings. Lecture Notes in Computer Science, vol. 372, pp. 1–17. Springer (1989). https://doi.org/10.1007/BFB0035748, https://doi.org/10.1007/BFb0035748
4. Anand, A., Mallik, K., Nayak, S.P., Schmuck, A.: Computing adequately permissive assumptions for synthesis. In: Sankaranarayanan, S., Sharygina, N. (eds.)

Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13994, pp. 211–228. Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_15, https://doi.org/10.1007/978-3-031-30820-8_15

5. Anand, A., Nayak, S.P., Schmuck, A.: Synthesizing permissive winning strategy templates for parity games. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13964, pp. 436–458. Springer (2023). https://doi.org/10.1007/978-3-031-37706-8_22, https://doi.org/10.1007/978-3-031-37706-8_22

6. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)

7. Basset, N., Raskin, J., Sankur, O.: Admissible strategies in timed games. In: Aceto, L., Bacci, G., Bacci, G., Ingólfsdóttir, A., Legay, A., Mardare, R. (eds.) Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 10460, pp. 403–425. Springer (2017). https://doi.org/10.1007/978-3-319-63121-9_20, https://doi.org/10.1007/978-3-319-63121-9_20

8. Belta, C., Yordanov, B., Gol, E.A.: Formal methods for discrete-time dynamical systems, vol. 89. Springer (2017)

9. Bernet, J., Janin, D., Walukiewicz, I.: Permissive strategies: from parity games to safety games. RAIRO Theor. Informatics Appl. 36(3), 261–275 (2002). https://doi.org/10.1051/ita:2002013

10. Berwanger, D.: Admissibility in infinite games. In: Thomas, W., Weil, P. (eds.) STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4393, pp. 188–199. Springer (2007). https://doi.org/10.1007/978-3-540-70918-3_17, https://doi.org/10.1007/978-3-540-70918-3_17

11. Beutner, R., Finkbeiner, B.: Prophecy variables for hyperproperty verification. In: 35th IEEE Computer Security Foundations Symposium, CSF 2022, Haifa, Israel, August 7-10, 2022. pp. 471–485. IEEE (2022). https://doi.org/10.1109/CSF54842.2022.9919658, https://doi.org/10.1109/CSF54842.2022.9919658

12. Bouyer, P., Markey, N., Olschewski, J., Ummels, M.: Measuring permissiveness in parity games: Mean-payoff parity games revisited. In: Bultan, T., Hsiung, P. (eds.) Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6996, pp. 135–149. Springer (2011). https://doi.org/10.1007/978-3-642-24372-1_11

13. Brenguier, R., Raskin, J., Sankur, O.: Assume-admissible synthesis. Acta Informatica 54(1), 41–83 (2017). https://doi.org/10.1007/S00236-016-0273-2, https://doi.org/10.1007/s00236-016-0273-2

14. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer Nature (2021)

15. Church, A.: Logic, arithmetic, and automata. Journal of Symbolic Logic 29(4), 210–210 (1964). https://doi.org/10.2307/2270398

16. Damm, W., Finkbeiner, B.: Automatic compositional synthesis of distributed systems. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014.

Proceedings. Lecture Notes in Computer Science, vol. 8442, pp. 179–193. Springer (2014). https://doi.org/10.1007/978-3-319-06410-9_13, https://doi.org/10.1007/978-3-319-06410-9_13

17. van Dijk, T.: Oink: An implementation and evaluation of modern parity game solvers. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10805, pp. 291–308. Springer (2018). https://doi.org/10.1007/978-3-319-89960-2_16, https://doi.org/10.1007/978-3-319-89960-2_16

18. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From spot 2.0 to spot 2.10: What's new? In: International Conference on Computer Aided Verification, CAV 2022. Lecture Notes in Computer Science, vol. 13372. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_9

19. Ehlers, R., Lafortune, S., Tripakis, S., Vardi, M.Y.: Supervisory control and reactive synthesis: a comparative introduction. Discret. Event Dyn. Syst. **27**(2), 209–260 (2017). https://doi.org/10.1007/s10626-015-0223-0

20. Finkbeiner, B.: Synthesis of reactive systems. In: Esparza, J., Grumberg, O., Sickert, S. (eds.) Dependable Software Systems Engineering, NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 45, pp. 72–98. IOS Press (2016). https://doi.org/10.3233/978-1-61499-627-9-72, https://doi.org/10.3233/978-1-61499-627-9-72

21. Finkbeiner, B., Metzger, N., Moses, Y.: Information flow guided synthesis. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13372, pp. 505–525. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_25, https://doi.org/10.1007/978-3-031-13188-2_25

22. Finkbeiner, B., Metzger, N., Moses, Y.: Information flow guided synthesis with unbounded communication. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14683, pp. 64–86. Springer (2024). https://doi.org/10.1007/978-3-031-65633-0_4, https://doi.org/10.1007/978-3-031-65633-0_4

23. Finkbeiner, B., Passing, N.: Synthesizing dominant strategies for liveness. In: Dawar, A., Guruswami, V. (eds.) 42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022, December 18-20, 2022, IIT Madras, Chennai, India. LIPIcs, vol. 250, pp. 37:1–37:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPICS.FSTTCS.2022.37, https://doi.org/10.4230/LIPIcs.FSTTCS.2022.37

24. Finkbeiner, B., Schmuck, A.K., Nayak, S.P., Metzger, N.: Artifact for synthesis of universal safety controllers at tacas 2025 (Jan 2025). https://doi.org/10.5281/zenodo.14627151, https://doi.org/10.5281/zenodo.14627151

25. Fremont, D.J., Seshia, S.A.: Reactive control improvisation. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification. pp. 307–326. Springer International Publishing, Cham (2018)

26. Goorden, M., Fabian, M.: No synthesis needed, we are alright already. In: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). pp. 195–202 (2019). https://doi.org/10.1109/COASE.2019.8843071

27. Jacobs, S., Pérez, G.A., Abraham, R., Bruyère, V., Cadilhac, M., Colange, M., Delfosse, C., van Dijk, T., Duret-Lutz, A., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, K.J., Michaud, T., Pommellet, A., Renkin, F., Schlehuber-Caissier, P., Sakr, M., Sickert, S., Staquet, G., Tamines, C., Tentrup, L., Walker, A.: The reactive synthesis competition (SYNTCOMP): 2018-2021. CoRR **abs/2206.00251** (2022). https://doi.org/10.48550/ARXIV.2206.00251, https://doi.org/10.48550/arXiv.2206.00251

28. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: Formal Methods in Computer-Aided Design, 6th International Conference, FMCAD 2006, San Jose, California, USA, November 12-16, 2006, Proceedings. pp. 117–124. IEEE Computer Society (2006). https://doi.org/10.1109/FMCAD.2006.22, https://doi.org/10.1109/FMCAD.2006.22

29. Jobstmann, B., Galler, S.J., Weiglhofer, M., Bloem, R.: Anzu: A tool for property synthesis. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4590, pp. 258–262. Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_29, https://doi.org/10.1007/978-3-540-73368-3_29

30. Kirsten, D.: Alternating tree automata and parity games. In: Grädel, E., Thomas, W., Wilke, T. (eds.) Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]. Lecture Notes in Computer Science, vol. 2500, pp. 153–167. Springer (2001). https://doi.org/10.1007/3-540-36387-4_9, https://doi.org/10.1007/3-540-36387-4_9

31. Klein, J., Baier, C., Klüppelholz, S.: Compositional construction of most general controllers. Acta Informatica **52**(4-5), 443–482 (2015). https://doi.org/10.1007/s00236-015-0239-9

32. Kress-Gazit, H., Lahijanian, M., Raman, V.: Synthesis for robots: Guarantees and feedback for robot behavior. Annual Review of Control, Robotics, and Autonomous Systems **1**(1), 211–236 (2018)

33. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. In: Halbwachs, N., Peled, D.A. (eds.) Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1633, pp. 172–183. Springer (1999). https://doi.org/10.1007/3-540-48683-6_17, https://doi.org/10.1007/3-540-48683-6_17

34. Latvala, T.: Efficient model checking of safety properties. In: Ball, T., Rajamani, S.K. (eds.) Model Checking Software, 10th International SPIN Workshop. Portland, OR, USA, May 9-10, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2648, pp. 74–88. Springer (2003). https://doi.org/10.1007/3-540-44829-2_5, https://doi.org/10.1007/3-540-44829-2_5

35. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32, https://doi.org/10.1109/SFCS.1977.32

36. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25**(1), 206–230 (Jan 1987). https://doi.org/10.1137/0325013, https://doi.org/10.1137/0325013
37. Schmuck, A., Moor, T., Majumdar, R.: On the relation between reactive synthesis and supervisory control of non-terminating processes. Discret. Event Dyn. Syst. **30**(1), 81–124 (2020). https://doi.org/10.1007/s10626-019-00299-5
38. Tabuada, P.: Verification and control of hybrid systems: a symbolic approach. Springer Science & Business Media (2009)
39. Wonham, W., Cai, K., Rudie, K.: Supervisory control of discrete-event systems: A brief history. Annual Reviews in Control **45**, 250–256 (2018)
40. Yin, X., Gao, B., Yu, X.: Formal synthesis of controllers for safety-critical autonomous systems: Developments and challenges. Annual Reviews in Control **57**, 100940 (2024). https://doi.org/https://doi.org/10.1016/j.arcontrol.2024.100940, https://www.sciencedirect.com/science/article/pii/S1367578824000099