Supervisory Controller Synthesis for Nonterminating Processes Is an Obliging Game

Rupak Majumdar and Anne-Kathrin Schmuck

Abstract—In this article, we present a new algorithm to solve the supervisory control problem over nonterminating processes modeled as \( \omega \)-regular automata. A solution to this problem was obtained by Thistle in 1995, which uses complex manipulations of automata. We show a new solution to the problem through a reduction to obliging games, which, in turn, can be reduced to \( \omega \)-regular reactive synthesis. Therefore, our reduction results in a symbolic algorithm based on manipulating sets of states.

Index Terms—Control systems, supervisory control, computer science.

I. INTRODUCTION

Supervisory control theory (SCT) is a branch of control theory, which is concerned with the control of discrete-event dynamical systems (DES) with respect to temporal specifications. Given such a DES, SCT asks to synthesize a supervisor that restricts the possible sequences of events such that any remaining sequence fulfills a given specification. The field of SCT was established by the seminal work of Ramadge and Wonham [1] concerning the control of terminating processes, i.e., systems whose behavior can be modeled by regular languages over finite words. This setting is well understood and summarized in standard textbooks [2], [3].

Already 30 years ago, Thistle and Wonham extended the scope of SCT to nonterminating processes [4], i.e., to the supervision of systems whose behavior can be modeled by regular languages over infinite words. Nonterminating processes naturally occur in models of infinitely executing reactive systems: \( \omega \)-words allow convenient modeling of both safety and liveness specifications for such systems. In a sequence of papers [4]–[6] culminating in [7], Thistle and Wonham laid out the foundations for SCT over nonterminating processes and showed, in particular, an algorithm to synthesize supervisors for general \( \omega \)-regular specifications under general \( \omega \)-regular plant properties. This symbolic synthesis algorithm involves an intricate fixed point computation over the \( \omega \)-regular languages, using structural operations on finite-state automata representations. As a direct result of the complexity of the involved operations, to the best of authors’ knowledge, this most general algorithm has never been implemented.

A key observation in Thistle and Wonham’s original work was the relationship between SCT and Church’s problem from logic [8], and hence to techniques from reactive synthesis. This influenced early works on using logical formalisms to control hybrid systems [9]–[11]. Unfortunately, the connection between SCT and reactive synthesis got mostly lost over time and is just about to be re-established. This article continues these recent connecting efforts [12]–[15].

While, conceptually, supervisor synthesis and reactive synthesis seem very similar, the resulting algorithmic reduction is not very obvious. To understand the source of difficulty, let us recall the setting of the problem. We are given a finite-state machine that forms the transition structure of the DES for the synthesis problem, and we are given two \( \omega \)-regular languages defined over this machine. The first language (let us call it \( A \)) models assumptions on the plant: a supervisor can assume that the (uncontrolled) plant language will satisfy this assumption. The second language (call it \( S \)) provides the specification that the supervisor must uphold whenever the plant operates in accordance to the assumptions, by preventing certain controllable events over time.

One can easily transform the given finite-state machine to a two-player game, as in reactive synthesis, and naively ask for a winning strategy for the winning condition \( A \Rightarrow S \), which states that if the plant satisfies its assumption, then the resulting behavior satisfies the specification. While this reduction seems natural, it is incorrect in the context of SCT. The problem is that a control strategy may “cheat” and enforce the aforementioned implication vacuously by actively preventing the plant from satisfying the assumption. In SCT, such undesired solutions are ruled out by a nonconflicting requirement: any finite word compliant with the supervisor must be extendable to an infinite word that satisfies \( A \). Hence, a nonconflicting supervisor always allows the plant to fulfill the assumption.

The main contribution of this article is a reduction of the supervisory control problem to a class of reactive synthesis problems called obliging games [16] that precisely capture a notion of nonconflicting strategies in the context of reactive synthesis. The main result of [16] shows that obliging games can be reduced to usual reactive synthesis on a larger game. Once the intuitive connection between supervisory control and obliging games is made, the formal reduction is almost trivial.

We consider this simplicity as a feature of our work: our conceptual reduction from supervisory control to obliging games, and hence to reactive synthesis, forms a separation of concerns between (a) the modeling of specifications and nonconflicting strategies and (b) the (nontrivial, but well-understood) algorithmics of solving games.

II. PRELIMINARIES

Formal Languages: Given a finite alphabet \( \Sigma \), we write \( \Sigma^*, \Sigma^+ \), and \( \Sigma^\omega \) for the sets of finite words, nonempty finite words, and infinite words over \( \Sigma \), and write \( \Sigma^\omega = \Sigma^* \cup \Sigma^\omega \). We call the subsets \( L \subseteq \Sigma^\omega \) and \( L \subseteq \Sigma^\omega \), a \( \omega \)-language and an \( \omega \)-language over \( \Sigma \), respectively. We write \( w \leq v \) (resp., \( w < v \)) if \( w \) is a prefix of \( v \) (resp., a strict prefix of \( v \)). The set of all prefixes of a word \( w \in \Sigma^\omega \) is a \( \omega \)-language denoted by \( \text{pf}(w) \subseteq \Sigma^\omega \). For \( L \subseteq \Sigma^\omega \), we have \( L \subseteq \text{pf}(L) \). A \( \omega \)-language \( L \) is prefix-closed if \( L = \text{pf}(L) \). The limit \( \lim(L) \) of a \( \omega \)-language \( L \) is the
Paths $x$ if for all $\exists$ alphabet $\Sigma$, initial state $x_0 \in X$, and partial transition function $\delta : X \times \Sigma \to 2^X$, for $x \in X$ and $\sigma \in \Sigma$, we write $\delta(x, \sigma)$ to signify that $\delta(x, \sigma)$ is defined. $M$ is deterministic if $\delta(x, \sigma)$ implies $|\delta(x, \sigma)| = 1$. A path of $M$ is a finite or infinite sequence $x = x_0x_1 \ldots$ s.t. for all $k \in \operatorname{Length}(x) - 1$ there exists some $x_k \in X$ s.t. $x_{k+1} \in \delta(x_k, \sigma_k)$. If $\pi$ is finite, we denote by $\operatorname{Last}(\pi) = \pi$ its last element. We collect all finite and infinite paths of $M$ in the sets $P(M) \subseteq x_0X^*$ and $P(M) \subseteq x_0X^\omega$, respectively. Given a string $s = \sigma_0\sigma_1 \ldots \in \Sigma^*$, we say that a path $\pi$ of $M$ is compliant with $s$ if $\operatorname{Length}(\pi) = \operatorname{Length}(s)$ and for all $k \in \operatorname{Length}(s) - 1$ we have $x_{k+1} \in \delta(x_k, \sigma_k)$. $\operatorname{Paths}_M(s)$ is the set of all paths of $M$ compliant with $s$. We collect all finite and infinite strings that are compliant with $M$ in the sets $L(M) := \{ s \in \Sigma^* \mid \operatorname{Paths}_M(s) \neq \emptyset \}$ and $L(M) := \{ s \in \Sigma^\omega \mid \operatorname{Paths}_M(s) \neq \emptyset \}$ respectively. If $M$ is deterministic we have $|\operatorname{Paths}_M(s)| = 1$ for all $s \in L(M)$. For a set of final states $F \subseteq X$, the tuple $(M, F)$ defines a deterministic finite-state automaton (DFA) over finite words. $(M, F)$ accepts the $\omega$-language $L(M, F)$ which contains all finite paths of $M$ which are ending in $F$. A $\omega$-language is called regular if there exists a DFA $(M, F)$ which accepts $L$.

Finite-state automata over infinite words: For a path $\pi$, define $\operatorname{Inf}(\pi) = \{ x \in X \mid x = x_1 \text{ for infinitely many } k \in \mathbb{N} \}$ to be the set of states visited infinitely often along $\pi$. We say that an infinite string $s \in \Sigma^\omega$ satisfies the Büchi condition $F^a = \{ F \}$ with $F \subseteq X$ on $M$ if there exists a path $\pi \in \operatorname{Paths}_M(s)$ such that $\operatorname{Inf}(\pi) \cap F \neq \emptyset$. Furthermore, let $F = \{ (G_1, R_1), \ldots, (G_m, R_m) \}$ be a set, s.t. $G_i, R_i \subseteq X$ for all $i = 1, \ldots, m$. A string $s \in \Sigma^\omega$ satisfies the Rabin condition $F^a = F$ on $M$ if there exists a path $\pi \in \operatorname{Paths}(M, s)$ such that $\operatorname{Inf}(\pi) \cap G_i \neq \emptyset$ and $\operatorname{Inf}(\pi) \cap R_i \neq \emptyset$ for some $i \in \{1; m\}$. It satisfies the Streett condition $F^a = F$ if $\operatorname{Inf}(\pi) \cap G_i = \emptyset$ or $\operatorname{Inf}(\pi) \cap R_i = \emptyset$ for all $i \in \{1; m\}$. Rabin and Streett conditions are dual; i.e., if $\pi$ satisfies $F^a$ it violates $F^a = F^a$.

We call a finite-state machine $M$ equipped with a Büchi, Rabin, or Streett acceptance condition $F$ a Büchi, Rabin, or Streett automaton, respectively, denoted by $(M, F)$. We collect all infinite strings (resp. paths) satisfying the specified acceptance condition $F$ over $M$, in the accepted language $L(M, F) \subseteq \Sigma^\omega$ (resp. in the set $P(M, F) \subseteq x_0X^\omega$). An $\omega$-language is called regular if it is accepted by a nondeterministic Büchi automaton. We remark that deterministic Rabin and deterministic Streett automata also accept precisely the set of regular languages. However, this is not true for deterministic Büchi automata, which are less expressive.

III. SUPERVISOR SYNTHESIS PROBLEM

We define the supervisory controller synthesis problem following the original formulation for $\omega$-languages [1] and the subsequent extension to $\omega$-languages in [4]–[7].

A. Problem Statement

Let $\Sigma$ be a finite alphabet of events partitioned into controllable events $\Sigma_C$ and uncontrollable events $\Sigma_{UC}$, that is, $\Sigma = \Sigma_C \cup \Sigma_{UC}$. A plant is a tuple $(L_P, L_P^\omega)$, where $L_P \subseteq \Sigma_C^\omega$ is a prefix-closed regular $\omega$-language and $L_P^\omega \subseteq \Sigma_C^\omega$ s.t. $\operatorname{pfx}(L_P^\omega) \subseteq L_P$ is a regular $\omega$-language. If, in addition, $\operatorname{pfx}(L_P) = L_P$, the plant is called deadlock-free. A specification is a tuple $(L_S, L_S^\omega)$ where $L_S \subseteq \Sigma_C^\omega$ is a regular $\omega$-language and $L_S^\omega := \operatorname{pfx}(L_S)$ is a prefix-closed regular $\omega$-language, that is deadlock-free.

Intuitively, the language-tuples $(L_P, L_P^\omega)$ and $(L_S, L_S^\omega)$ capture both safety and liveness properties. Here, $(L_P, L_P^\omega)$ models the properties the uncontrolled plant exhibits. In contrast, $(L_S, L_S^\omega)$ restricts the behavior of the plant to a set of desired behaviors which is, by definition, deadlock-free. That is, every safe event sequence generated by the plant under control must be extendable to an infinite string additionally fulfilling the imposed liveness requirements. The set $\Sigma_C$ denotes all events the controller can prevent the plant from executing, while the set $\Sigma_{UC}$ denotes events that cannot be prevented by the controller.

A control pattern $\gamma$ is a subset of $\Sigma$ containing $\Sigma_{UC}$. We collect all control pattern in the set $\Gamma := \{ \gamma \mid \gamma \subseteq \Sigma_C \subseteq \gamma \}$. Given this set, a (string-based) supervisor is defined as a map $f : \Sigma^\omega \to \Gamma$ that maps each (finite) past event sequence $s \in \Sigma^*$ to a control pattern $f(s) \in \Gamma$. The control pattern specifies the set of enabled successor events after the occurrence of $s$. The definition of control patterns ensures that uncontrollable events are always enabled. A word $s \in \Sigma^*$ is called consistent with $f$ if for all $\sigma \in \Sigma$ and $\tau \in \operatorname{pfx}(s)$, it holds that $\sigma \in f(\tau)$. We write $L_f$ for the set of all words consistent with $f$ and define $L_f := \operatorname{lim}(L_f)$.

With these definitions, the supervisor synthesis problem can be formally stated as follows.

Problem 1 (String-Based Supervisor Synthesis): Given an alphabet $\Sigma = \Sigma_C \cup \Sigma_{UC}$, a plant model $(L_P, L_P^\omega)$, where $L_P \subseteq \Sigma_C^\omega$ and $\operatorname{pfx}(L_P^\omega) \subseteq L_P \subseteq \Sigma_C^\omega$ are regular languages, and a regular specification language $\Sigma_{UC} \subseteq \Sigma^\omega$, synthesize, if possible, a string-based supervisor $f : \Sigma^\omega \to \Gamma$ s.t. both

i) the closed-loop satisfies the specification, i.e.,
$$\emptyset \subseteq L_f \cap L_P^\omega \subseteq L_S$$  \hspace{1cm} (1a)

ii) the plant and the supervisor are nonconflicting, i.e.,
$$L_f \cap L_P \subseteq \operatorname{pfx}(L_f \cap L_P)$$  \hspace{1cm} (1b)

or determine that no such supervisor exists. A string-based supervisor solves the synthesis problem over $(L_P, L_P^\omega), L_S$ if it satisfies (1a) and (1b).

Constraint (1b) ensures that the plant is always able to generate events allowed by $f$ s.t. it ultimately generates a word in the language $L_P$. Then, by (1a), all such generated words must be contained in the specification $L_S$.

B. Automata Representations for Supervisor Synthesis

As the given plant and specification languages are assumed to be regular, we know that each tuple $(L_\alpha, L_{\alpha^\omega})$ with $\alpha \in \{P, S\}$ can be represented by a deterministic finite-state machine $M_\alpha$ equipped with a Robin or Streett acceptance condition $F_\alpha$ s.t. $L_\alpha = L(M_\alpha)$ and $L_{\alpha^\omega} = L(M_{\alpha^\omega})$.

This actually coincides with the basic setting of SCT for terminating processes, where the language $L_\alpha$ is a nonprefix closed $\omega$-language, rather than an $\omega$-language and each tuple $(L_\alpha, L_{\alpha^\omega})$ is, therefore, realized by a DFA instead. A standard algorithmic solution to Problem 1 for terminating processes combines the plant and specification realizations

\footnote{1In language theory, an $\omega$-language $L$ is a safety language if $\operatorname{lim}(\operatorname{pfx}(L)) = L$, and a liveness language if $\lim(\operatorname{pfx}(L)) = \Sigma^\omega$. Hence, $\lim(L_P)$ and $\lim(L_S)$ are safety languages. Furthermore, any $\omega$-regular language can be written as the intersection of a safety and a liveness language [17]. Therefore, there exist liveness languages $L_P$ and $L_S$ s.t. $L_P = \emptyset$ and $L_S = \emptyset \cap \lim(L_P)$ and $L_S = L_S \cap \lim(L_P)$ are both safety and liveness properties.

\footnote{2As $L_S := \operatorname{pfx}(L_S)$, the language $L_S$ is uniquely determined by $L_S$ and, therefore, omitted from the problem description.}
into a single DFA \((M, F)\). This automaton is then manipulated in various steps to compute the supervisor \(f\) solving Problem 1.

In a very similar manner,\(^3\) two \(\omega\)-automata \((M_P, F_P)\) and \((M_S, F_S)\) can be combined into a finite-state machine \(M\) equipped with two sets of marked states \(F_P\) and \(F_S\) over \(M\). This machine similarly serves as an input to an automaton-based supervisor synthesis procedure for nonterminating processes and is formalized as follows.

**Proposition III.1** (See [5], [7], [18]): Let \(L_P \subseteq \Sigma^\omega\), \(L_P, L_S \subseteq \Sigma^\omega\) be regular languages and \(((L_P, L_P), L_S)\) an input to Problem 1. Then, there exists a finite-state machine \(M = (X, \Sigma, \delta, x_0)\), a Street condition \(\mathcal{F}_P\) over \(M\) and a Rabin condition \(\mathcal{F}_S\) over \(M\), s.t.

1. \(L_P = L(M)\);
2. \(L_P = L(M, F_P)\);
3. \(L_S \cap L(M) = L(M, F_S)\);
4. \(M\) is deterministic,
5. distinct transitions in \(M\) carry distinct labels, i.e., for any \(\sigma, \sigma' \in \Sigma\) and \(x \in X\), we have that \(\delta(x, \sigma) \neq \delta(x, \sigma')\) implies \(\sigma = \sigma'\).

**Definition III.1:** If conditions (a)–(e) in Proposition III.1 are fulfilled, we call the tuple \((M, F_P, F_S)\), the Street/Rabin supervisor synthesis automaton realizing \(((L_P, L_P), L_S)\).

We remark that the conditions imposed on \(M\) in Proposition III.1 are without loss of generality. Conditions (a)–(d) follow from the fact that any omega-regular language can be realized by a deterministic Rabin or Street automaton, which can be extended to generate the respective unmarked languages. Then, the resulting automata can be combined by a product construction to obtain \((M, F_P, F_S)\). Finally, regarding condition (e), an automaton with nondistinct labels can be modified by extending the state space \(X\) to \(X \times \Sigma\) to fulfill this property.

### C. Path-Based Supervisor Synthesis

We now turn to effective algorithms for supervisor synthesis when the input \(((L_P, L_P), L_S)\) to Problem 1 is realized by a Street/Rabin supervisor synthesis automaton \((M, F_P, F_S)\). This allows us to reformulate Problem 1 in terms of \((M, F_P, F_S)\) and so called path-based supervisors. A path-based supervisor is a map \(f : (P(M) \to \Gamma)\) of a path \(\pi\) over \(M\) is called consistent with \(f\) if for all \(x, x' \in X\) and \(\nu \in X^\ast\) s.t. \(\nu x \nu x' \in \operatorname{pfx}(\pi)\), there exists an event \(\sigma \in f(\nu x)\) s.t. \(x' = \delta(x, \sigma)\). Let \(P(M, f)\) be the set of all paths of \(M\) consistent with \(f\) and define \(P(M, f) := \lim(P(M, f))\). We can now restate Problem 1 into the following path-based supervisor synthesis problem.

\textbf{Problem 2 (Path-based Supervision):} Given a Street/Rabin supervisor synthesis automaton \((M, F_P, F_S)\), synthesize, if possible, a path-based supervisor \(f : (P(M) \to \Gamma)\).

\begin{align*}
\emptyset & \subseteq P(M, f) \cap P(M, F_P) \subseteq P(M, F_S), \quad \text{(2a)} \\
\text{P}(M, f) & \subseteq \text{pfx}(P(M, f) \cap P(M, F_P)) \quad \text{(2b)}
\end{align*}

or determine that no such supervisor exists. A path-based supervisor \(f\) solves the synthesis problem over \((M, F_P, F_S)\) if it satisfies (2a) and (2b).

The structure of the finite-state machine \(M\) ensures that there is a one-to-one correspondence between a word in \(L_P = L(M)\) and its unique path \(\pi = \text{Paths}_M(s)\) over \(M\). Furthermore, as transition labels are unique in \(M\) (see Proposition III.1 (e)), there is also a unique word \(s\) associated with a path \(\pi\) over \(M\). With these observations, we can show that Problems 1 and 2 are indeed equivalent, as summarized by Theorem III.2.

\[\forall s \in L(M) \quad f(s) = f(\text{Paths}_M(s)).\]

Then, \(f\) solves the synthesis problem over \(((L_P, L_P), L_S)\) iff \(f\) solves the synthesis problem over \((M, F_P, F_S)\).

\[\text{Proof:}\] We prove one direction. The other one follows from the same reasoning. Fix \(f\) s.t. (1) holds, and \(f\) s.t. (3) holds.

\begin{itemize}
\item \(\text{Show } P(M, f) \cap P(M, F_P) \neq \emptyset.\)
\item \(\text{Show } \text{P}(M, f) = \text{pfx}(P(M, f) \cap P(M, F_P)).\)
\end{itemize}

\[\text{Algorithms solving various versions of Problem 2 are studied by Thistle and Wonham in [4]–[7]. All of them are initialized with a deterministic finite-state machine } M \text{ equipped with two acceptance conditions } F_P \text{ and } F_S \text{ where } F_S \text{ is a Rabin condition. However, } F_P \text{ is chosen to be trivial in [4] (i.e., } L_P = \Sigma^\omega\text{), a deterministic Büchi condition in [5] and [6], and a deterministic Streett condition in [7], i.e., the algorithm in [7] solves Problem 2. In the remaining sections of this article, we show an alternative way to solve Problem 2, which establishes a new connection between the fields of supervisory control and reactive synthesis.}\]

### D. Simple Example

Consider the finite-state machine \(M\) depicted in Fig. 1 for a path-based supervisor synthesis problem. Here, the alphabet is \(\Sigma = \{a, b, c\}.\)
partitioned in $\Sigma_c = \{a, b\}$ (indicated by a tick on the corresponding edges in Fig. 1) and $\Sigma_u = \{c\}$. The uncontrolled plant is assumed to only generate traces allowed in $M$ (safety) and to additionally visit the state $p_2$ always again (liveness). The latter is modeled by a Büchi acceptance condition $F^B_p = \{p_2\}$, and indicated by the light-blue double circle around state $p_2$ in Fig. 1. The Büchi condition $F^B_p$ can be equivalently formulated as the Streett condition $F^S_p = \{(p_0, p_1, p_2), \{p_2\}\}$.

The specification requires that the controlled plant should visit state $p_1$ always again (liveness) and does not dead-lock (safety). This can be modeled by a Büchi condition with $F^B_p = \{p_1\}$, indicated by the red double circle around $p_1$ in Fig. 1. Again, we can equivalently represent $F^B_p$ as the Rabin condition $F^R_p = \{(p_1), \emptyset\}$. In order to achieve the desired specification, the supervisor can only disable controllable actions; thus, every control pattern allows $c$.

The supervisor synthesis problem, Problem 2, now asks to synthesize a path-based supervisor that ensures (i) whenever $p_2$ is always visited again, also $p_1$ is always visited again, and that (ii) the controller never prevents the plant from visiting $p_2$ again in the future. A path-based supervisor solving this is given by the following rule: any path ending in $p_0$ is mapped to $\{a, c\}$, any path ending in $p_1$ is mapped to $\{b, c\}$, and any path ending in $p_2$ is mapped to $\{a, c\}$. This effectively disables the self-loop on event 6 in state $p_2$.

IV. FROM SUPERVISOR SYNTHESIS TO GAMES

We shall reduce Problem 2 to solving a class of two-player games on graphs with $\omega$-regular winning conditions.

A. Two-Player Games

A two-player game graph $G = (Q^0, Q^1, \delta^0, \delta^1, q_{init})$ consists of two finite disjoint state sets $Q^0$ and $Q^1$, two transition functions $\delta^0 : Q^0 \to 2^{Q^1}$ and $\delta^1 : Q^1 \to 2^{Q^0}$, and an initial state $q_{init} \in Q^0$. We write $Q = Q^0 \cup Q^1$. Given a game graph $G$, a strategy for player 0 is a function $h : q_{init}(Q^1)^* \to Q^1$. The sequence $\rho \in Q^\infty$ is called a play over $G$ if $\rho(0) = q_{init}$ and for all $k \in \text{Length}(\rho) - 1$, we have $\rho(k + 1) \in \delta^1(\rho(k))$ if $\text{Last}(\rho) \in Q^0$ and $\rho(k + 1) \in \delta^1(\rho(k))$ otherwise. The play $\rho$ is compliant with $h$ if additionally $h(\rho[p_{0}, k]) = p_{0}(k + 1)$ if $\text{Last}(\rho) \in Q^0$. We denote by $P(G, h)$ and $\mathcal{P}(G, h)$, the set of finite and infinite plays over $G$ compliant with $h$.

We define $\omega$-regular winning conditions for two-player games. These are specified analogously to acceptance conditions for finite-state machines over subsets of states $Q$. That is, we consider Büchi, Rabin and Streett conditions $F$ as defined in Section II over subsets of $Q$ and say that a play $\rho$ is winning w.r.t. $F$ if $\rho$ satisfies $F$ on $G$. In addition, we also consider the parity accepting condition [19]. For the parity condition with $k$ priorities, we assume there is a coloring function $\Omega : Q \to \{0, \ldots, k - 1\}$. A play $\rho$ is winning if the maximum color seen infinitely often is even.

We call a game graph equipped with a Büchi, Rabin, Streett, or parity winning condition $F$ a Büchi, Rabin, Streett, or parity game, respectively, and denote it by the tuple $(G, F)$. The set of all winning plays over $G$ w.r.t. $F$ is denoted $\mathcal{P}(G, F)$. A strategy $h$ is winning in a game $(G, F)$, if $\mathcal{P}(G, h) \subseteq \mathcal{P}(G, F)$. We remark that it is decidable if player 0 has a winning strategy in a two-player game with a Büchi, Rabin, Streett, or parity winning condition [19]–[22].

B. Supervisor Synthesis as a Two-Player Game

Intuitively, one can interpret the interaction of a supervisor with the plant as a two-player game over $M$. Player 0 (the supervisor) picks a control pattern $\gamma \in \Gamma$ and player 1 (the plant) resolves the remaining nondeterminism by choosing a transition allowed by $\gamma$. We formalize the construction as follows.

Definition IV.1: Let $M = (X, \Sigma, \delta, q_0)$ be as in Proposition III.1 with $\Sigma_{uc} \subseteq \Sigma$ and $\Gamma := \{\gamma \subseteq \Sigma \mid \Sigma_{uc} \subseteq \gamma\}$. Then, we define its associated game graph as $G(M) = (Q^0, Q^1, \delta^0, \delta^1, q_{init})$ s.t.:

a) $Q^0 = X$;

b) $Q^1 = X \times \Gamma$;

c) $\delta^0(x) = \{x\} \times \Gamma$;

d) $x' \in \delta^1((x, \gamma))$ iff $\sigma \in \gamma$ and $x' = \delta(x, \sigma)$.

Intuitively, the game graph $G$ makes the choice of the control pattern taken by the state-based supervisor over $M$ explicit by inserting player 1 states in between any two player 0 states, i.e. the choice of control pattern $\gamma$ in state $x \in X$ of $M$ corresponds to the move of player 0 from $q = x$ to $q' = (x, \gamma)$ in $G$. Furthermore, as $M$ is assumed to have unique transition labels, this expansion allows us to remove all transition labels resulting in an unlabeled game graph $G$ as defined in Section IV-A. Fig. 2 shows the two-player game graph $G(M)$ corresponding to $M$ in Fig. 1.

We now discuss an appropriate winning condition for the game. Consider the state-based supervisor synthesis problem (Problem 2) over the Streett/Rabin supervisor synthesis automaton $(M, F^S_p, F^R_p)$. Here, (2a) requires that any infinite trace over $M$, which is both compliant with $f$ and fulfills the plant assumption $\mathcal{L}_P$ also fulfills the specification $\mathcal{L}_S$. Hence, we can equivalently write (2a) as the implication

$$\forall \pi \in \mathcal{P}(M, f) \cdot (\pi \in \mathcal{P}(M, F^S_p) \Rightarrow \pi \in \mathcal{P}(M, F^R_p)).$$

which is in turn equivalent to

$$\forall \pi \in \mathcal{P}(M, f) \cdot (\pi \notin \mathcal{P}(M, F^S_p) \lor \pi \in \mathcal{P}(M, F^R_p)).$$

Consequently, (2a) is achieved by a supervisor $f$, which ensures plays over $M$ either do not satisfy the Streett condition $F^S_p$, or fulfill the Rabin condition $F^R_p$. However, as Rabin and Streett conditions are duals, not satisfying the Streett condition $F^S_p$ is equivalent to satisfying the Rabin condition $F^R_p$. Consequently, given the definition of Rabin winning conditions (see Section II), it is easy to see that a path over $M$ satisfies either the Rabin condition $F^R_p$ or the Rabin condition $F^R_{p \upharpoonright S}$ if it satisfies the Rabin condition $F^R_p \upharpoonright S = F^R_p \upharpoonright S$. With this observation, we can further rewrite (2a) into the equivalent formula

$$\mathcal{P}(M, f) \subseteq \mathcal{P}(M, F^R_p \upharpoonright S).$$

Thus, an obvious choice for the winning condition over the game graph $G(M)$ is the Rabin condition $F^R_p \upharpoonright S$.

Example IV.1: Consider the example from Section III-D and recall that $F^S_p = \{(p_0, p_1, p_2), \{p_2\}\}$ and $F^R_p = \{(p_1), \emptyset\}$. This gives
the Rabin winning condition
\[ F_P^{\nu,S} = \{ (\{ p_0, p_1, p_2 \}, \{ p_2 \}), (\{ p_1 \}, \emptyset) \} \quad (7) \]

for the induced game over \( G(M) \). Intuitively, the condition in (7) states that either \( p_2 \) is only visited \textit{finitely often} (first Rabin pair) or \( p_1 \) is visited \textit{infinitely often} (second Rabin pair). These two possibilities admit winning strategies that either prevent the plant from fulfilling its liveness properties (e.g., by always disabling \( a \) and \( b \) in all states) or that ensure that the specification gets fulfilled (e.g., by choosing the strategy given in Section III-D).

As the abovementioned example demonstrates, a winning strategy for \( F_P^{\nu,S} \) may not fulfill condition (2b). A strategy can choose to satisfy (7) vacuously, by actively preventing the plant to fulfill its liveness properties. Thus, we need to modify the winning condition to ensure the resulting strategy satisfies both (2a) and (2b). As the nonconflicting requirement of (2b) is not a linear property [12], it cannot be easily “compiled away” in reactive synthesis. Therefore, we consider a different type of game instead, called obliging game.

V. SUPERVISOR SYNTHESIS VIA OBLIGING GAMES

A. Obliging Games

An obliging game [16] is a triple \( (G, S, W) \) where \( G \) is a game graph and \( S \) and \( W \) are two winning conditions, called strong and weak, respectively. To win an obliging game, player 0 (the “controller”) needs to ensure the strong winning condition \( S \) against any strategy of player 1 (the “system”), while allowing the system to cooperate with him to additionally fulfill \( W \). Such winning strategies are, therefore, called \textit{gracious} and the synthesis problem for obliging games asks to synthesize such a gracious control strategy or determine that none exists, as formalized in the following problem statement.

Problem 3 (Obliging Games): Given an obliging game \((G, S, W),\)

\( i \) every play over \( G \) compliant with \( h \) is winning w.r.t. \( S \)
\[ \mathcal{P}(G, h) \subseteq \mathcal{P}(G, S) \quad (8a) \]

\( ii \) for every finite play \( \nu \) over \( G \) compliant with \( h \), there exists an infinite play \( \rho \) over \( G \) compliant with \( h \) and winning w.r.t. \( W \) s.t. \( \nu \in \text{pfix}(\rho) \)
\[ \mathcal{P}(G, h) \subseteq \text{pfix}(\mathcal{P}(G, h) \cap \mathcal{P}(G, W)) \quad (8b) \]

or determine that no such strategy exists.

The following theorem characterizes the solution of Problem 3 by a reduction to a parity game. As parity games are decidable and one can effectively construct winning strategies of player 0 in such games, Theorem V.1 establishes that the same is true for obliging games.

Theorem V.1: Every obliging game \((G, S, W)\) is reducible to a two-player game with an \( \omega \)-regular winning condition. In particular, an obliging game \((G, F^p, F^s)\) with \( n \) states, a Rabin condition \( F^p \) with \( k \) pairs, and a Streett condition \( F^s \) with \( \ell \) pairs can be reduced to a two-player game with \( nk \ell 2^{O(n)} \) states, a parity condition with \( 2k + 2 \) colors, and \( 2k \ell 2^{O(n)} \) memory.

Proof: The first claim follows from the construction of a Streett condition with \( \ell \) pairs, one can construct a (nondeterministic) Büchi automaton with \( 2^{O(n)} \) states that accepts the same language. Moreover, by taking a product with a monitor with \( k^2 \cdot k! \) states, we can convert the Rabin condition to a parity condition [19] with \( 2k \) colors. Now, the construction in [16, Lem.2, Thm.4] reduces an obliging game with \( n \) states, a strong parity winning condition with \( 2k \) colors and a weak winning condition accepted by a Büchi automaton with \( q \) states into a game with \( O(nq) \) states, \( 2k + 2 \) colors, and memory \( 2k \ell 2^{O(n)} \).

In order to reduce the supervisor synthesis problem to obliging games we need to define appropriate winning conditions. We can see by inspection that after replacing (2a) by (6) in Problem 2 and defining \( S := F_P^{\nu,S} \) and \( W := F_P^S \), in Problem 3, the two problem descriptions match. However, the system models and the corresponding control mechanisms are different. We, therefore, need to match path-based supervisors for \( M \) with player 0 strategies over \( G(M) \).

B. Formal Reduction

Given the reduction from \( M \) to a game graph \( G(M) \), and the strong and weak winning conditions, it remains to show that the resulting obliging game is indeed equivalent to the path-based supervisor synthesis problem. This is formalized in the following theorem.

Theorem V.2: Let \((M, F_P^p, F_P^S)\) be a Streett/Rabin supervisor synthesis automaton and \( G(M) \) its associated game graph. Then, there exists a path-based supervisor \( f \) that is a solution to the supervisor synthesis problem over \((M, F_P^p, F_P^S)\) if there exists a player 0 strategy \( h \) winning the obliging game \((G(M), F_P^{\nu,S}, F_P^S)\).

In order to prove Theorem V.2, we first formalize a mapping from paths over \( M \) to plays over \( G \) and back. This will allow us to define corresponding path-based supervisors and gracious strategies and formalize their associated properties.

Paths versus Plays: To formally connect paths in \( M \) to plays over \( G \), we define the set-valued map \( \text{Plays} : x_0 X^* \rightarrow 2^{(nQ)^Q} \), iteratively as follows:
\[ \text{Plays}(x_0) := \{ x_0 \} \text{ and } \text{Plays}(x\nu) := \{ \mu \nu | \mu \in \text{Plays}(\nu), x \in (\text{Last}(\nu) \times \Gamma) \}. \]

By slightly abusing notation, we extend the map \( \text{Plays} \) to infinite paths \( x \in x_0 X^* \) as the limit of all mappings \( \text{Plays}(p) \) where \( p \in x_0 X^* \) is the unbounded monad sequence of prefixes of \( x \). Similarly, we define the inverse map \( \text{Plays}^{-1} : (Q^Q) \rightarrow x_0 X^* \), \( \text{Plays}^{-1}(\mu) = \nu \) where \( \nu \) is the single element of the set \( \{ \nu \in x_0 X^* | \mu \in \text{Plays}(\nu) \} \). Again we extend \( \text{Plays}^{-1} \) to infinite strings in the obvious way.

The construction of \( G(M) \) from Definition IV.1 allows us to show that the maps \( \text{Plays} \) indeed captures all the information required to map paths over \( M \) to the corresponding plays over \( G(M) \) and vice versa.

Lemma V.3: Let \( M \) be a finite-state machine as in Proposition III.1 and \( G(M) \) its associated game graph as in Definition IV.1. Then
\[ \text{Plays}(P(M)) = P(G), \quad (9a) \]
\[ \text{Plays}(P(M)) = P(G), \quad \text{and} \]
\[ \text{Plays}(P(M, F)) = P(G, F), \quad (9c) \]

where \( F \) is a winning condition over \( M \).

Proof:
\[ \Box \]
\[ \text{(9a)} \]: \( \nu = x_0 x_1 \ldots x_k \in P(M) \). Then, \( \text{Plays}(\nu) \) is the set containing all plays \( \mu \in \nu x_0 \gamma_0 \gamma_1 x_1 \gamma_1 \ldots (x_k \gamma_k x_k) \in \nu \times [0; k] \). \( \ell \) follows from Definition IV.1 that all \( \mu \in \text{Plays}(\nu) \) are a play over \( G \) starting in \( q_0 \) and, hence \( \text{Plays}(P(M)) \subseteq P(G) \). The inverse direction follows similarly from the last condition in Definition IV.1.

\[ \Box \]
\[ \text{(9b)} \] follows directly from (9a) by taking the limit closure on both sides.

\[ \Box \]
\[ \text{(9b)} \] First, any winning condition over \( M \) is also a winning condition over \( G \) as \( Q = Q^Q \cup Q^Q \) with \( Q^Q \). Now pick any path \( \pi \in P(M, F) \). Then, we know that the set \( \text{Inf}(\pi) \subseteq X \) fulfills the conditions for acceptance w.r.t. the acceptance condition \( F \) over \( M \). Now take any \( \rho \in \text{Plays}(\pi) \subseteq P(G) \) and observe that deciding winning of \( \rho \) w.r.t. \( F \subseteq Q^Q \) only depends on the set \( \text{Inf}(\rho)|_{Q^Q} \subseteq Q^Q \).

IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 68, NO. 1, JANUARY 2023
Supervisors versus Strategies: Unfortunately, we cannot directly utilize the properties in (9) to relate path-based supervisors and gracious strategies. By definition, control strategies can base their decision on all information from the past observed state sequence. As one path over M corresponds to multiple plays over $G(M)$, such every play could in principle induce a different control decision. We call strategies that do not utilize this additional flexibility nonambiguous.

Definition V.1: Let $G$ be as in Definition IV.1. We call a player 0 strategy over $G$ nonambiguous if for any $\nu \in x_0X^*$ and any $\mu, \mu' \in \text{Plays}(\nu)$, we have $h(\mu) = h(\mu')$.

A strategy over $G$ can only choose one particular next state in the current one. As the initial state is unique, there must be a unique control pattern chosen in this state leading to a unique next state in $G$. Iteratively applying this argument shows that there is a unique play over $G$ generated under any control strategy $h$. Therefore, we can always construct a nonambiguous strategy $\hat{h}$ over $G$ from a given control strategy $h$ with the same set of generated plays.

Proposition V.4: Given the premises of Definition V.1, let $h$ be a strategy over $G$, then $\hat{h}$ s.t.

$$\hat{h}(x_0) := hx_0, \quad \hat{h}(\mu x_k x_{k+1}) := h\mu h(\mu) x_{k+1}$$

(10)

is a nonambiguous player 0 strategy over $G$ and it holds that $\mathcal{P}(G, h) = \mathcal{P}(\hat{G}, \hat{h})$.

Proof: For the base-case, $\text{Plays}(x_0) = \{x_0\}$ and, therefore, for all $\mu, \mu' \in \text{Plays}(x_0)$ we have $\mu = \mu' = x_0$. Hence, $h(\mu) = h(\mu') = hx_0$, i.e., $\mathcal{P}(G, h)_{[0,0]} = \mathcal{P}(\hat{G}, \hat{h})_{[0,0]}$. For the induction step, fix $\nu \in x_0X^*$ with $|\nu| = k > 1$ and assume that for all $\mu, \mu' \in \text{Plays}(\nu)$ we have $h(\mu) = h(\mu')$. Now choose any $x \in X$ and observe that $\text{Plays}(x\nu) = \{\mu x | \mu \in \text{Plays}(\nu), x \in \{\text{Last}(\nu)\} \times X\}$. Now pick any two $\mu x, \mu' x \in \text{Plays}(x\nu)$ and observe that from the definition of $h$ follows that $h(\mu x) = h\mu h(\mu)x$ and $h(\mu' x) = h\mu' h(\mu')x$. As $\mu, \mu' \in \text{Plays}(\nu)$ it follows from the induction hypothesis that $h(\mu) = h(\mu')$ and, therefore, $h(\mu x) = h(\mu' x)$. This proves that $h$ is nonambiguous. Now assume $A := \mathcal{P}(G, h)_{[0,k]} = \mathcal{P}(\hat{G}, \hat{h})_{[0,k]}$ and $\hat{h}(\mu) = h\mu h(\mu)$ for all $\mu \in A$. Then, it follows from Definition IV.1 that $\mathcal{P}(G, h\mu)_{[0,k+1]}$ contains all strings $\mu \text{Last}(\mu), \gamma x'$. s.t. $\mu \in A$. $\text{Last}(\mu), \gamma \in \{\mu\}$ and $x' = \delta(\sigma, x)$ for some $\sigma \in \gamma$. With this it immediately follows from the induction hypothesis that $\mathcal{P}(G, h\mu)_{[0,k+1]} = \mathcal{P}(\hat{G}, \hat{h})_{[0,k+1]}$. As both $\mathcal{P}(G, h)$ and $\mathcal{P}(\hat{G}, \hat{h})$ are closed languages, this proves the claim.

Proposition V.4 shows that restricting attention to nonambiguous player 0 strategies over $G$ is without loss of generality. Now it is easy to see that nonambiguous strategies over $G(M)$ allow for a one-to-one correspondence with path-based supervisors over $M$, which finally leads to the desired correspondence between Problem 2 and Problem 3.

Proposition V.5: Given the premises of Theorem V.2 the following holds.

i) Let $\bar{f}$ be a supervisor solving $(M, \mathcal{F}_p^3, \mathcal{F}_P^3)$ and $\bar{h}$ a player 0 winning strategy over $G(M)$ s.t.

$$\forall \mu \in q_0(Q^1Q^0)^* \cdot h(\mu) = (\text{Last}(\mu), \bar{f}(\text{Plays}^{-1}(\mu))).$$

Then, $\bar{h}$ is a nonambiguous winning strategy for $(G(M), \mathcal{F}_P^0, \mathcal{F}_P^3)$. 

ii) Let $\hat{h}$ be a nonambiguous winning strategy for $(G(M), \mathcal{F}_P^0, \mathcal{F}_P^3)$ and $f$ s.t.

$$\hat{f}(\nu) = \gamma \text{ with } \gamma \in \{\exists \mu \in \text{Plays}(\nu) \cdot \hat{h}(\mu) = (\cdot, \gamma)\}.$$ 

(12)

Then, $\hat{f}$ is a path-based supervisor solving $(M, \mathcal{F}_p^0, \mathcal{F}_P^3)$.

Proof: First, observe that given $\bar{f}$, every $\hat{f}$ fulfilling (11) is nonambiguous by construction. Conversely, given a nonambiguous strategy $\hat{h}$, (12) implies that $\gamma$ is uniquely defined for any $\nu \in x_0X^*$, i.e., $\hat{f}$ is a path-based strategy over $M$. Furthermore, given nonambiguity of $\hat{h}$ we can combine the induction from the proof of Proposition V.4 and the correspondence used in the proof of (9a) to conclude that

$$\mathcal{P}(G, \hat{h}) = \mathcal{P}(\mathcal{M}, \bar{f}).$$

(13)

Now assume (2a) [equivalently (6)] holds for $\bar{f}$. As the map $\mathcal{P} \circ \mathcal{P}$ is monotonous, this gives $\mathcal{P}(\mathcal{M}, \bar{f}) \subseteq \mathcal{P}(\mathcal{M}, \mathcal{S})$. Then, it follows from (13) and (9c) that (2a) implies that (8a) holds for $\hat{h}$. Now assume (2b) holds for $\bar{f}$ and observe that the map $\mathcal{P} \circ \mathcal{P}$ fulfills the following properties: (a) $\mathcal{P}(\text{Plays}(A)) = \mathcal{P}(\mathcal{P}(\text{Plays}(A)))$, and (b) $\mathcal{P}(A \cap B) = \mathcal{P}(\mathcal{P}(A \cap \mathcal{P}(B))).$ With this, it follows from (13) and (9c) that (2b) implies that (8b) holds for $\hat{h}$.

The reverse direction follows from the same reasoning and is, therefore, omitted.

With this, we see that Theorem V.2 is an immediate corollary of Propositions V.4 and V.5.

C. Example

The technical reduction from obliging games to games with $\omega$-regular winning conditions (see Theorem V.1) can be found in [16]. We give an intuitive explanation of this construction by applying it to our example and thereby constructing a winning strategy for the obliging game $(G(M), \mathcal{F}_P^0, \mathcal{F}_P^3)$ over the game graph $G(M)$ depicted in Fig. 2.

As the first step of this construction, we double the state space of $G$ resulting in an upper and a lower part (see Fig. 3). The upper part is a copy of the old state space while in the lower part all states become control player states (indicated by their violet ellipse shape). Now we run the following Gedankenexperiment: in every (rectangular green) plant state the plant can choose between deciding on the next executed event by herself or allowing the controller to make this choice for her. In the first case, the play stays within the upper part (using a dashed green transition), while in the second case the play moves to the lower part (using a dotted orange transition) and the controller decides the next move on behalf of the plant (by taking an available solid violet transition). In each case, the play moves to a control player’s state ($p_i$).
(top) or \(p'_i\) (bottom), with \(i \in \{0, 1, 2\}\). In both cases, the controller chooses a control pattern \(\gamma\) and by this always moves to the rectangular green state \((p_i, \gamma)\) in the upper part. Here, it is again the choice of the plant to either stay in the original (top) game or to move to the bottom copy.

With this modified game in mind, we can interpret the two copies of the game graph as follows. In the top one, the controller is only concerned with fulfilling the specification, i.e., solving a standard two-player game with the winning condition \(F^p\) in (7).

The bottom copy of the game makes sure that the resulting strategy is nonconflicting. Within the outlined Gedankenexperiment, this is ensured by the fact that at any point in time, the plant can decide to hand over all future choices of the next events to the controller and the controller must be able to demonstrate that the liveness condition of the plant (i.e., \(F^p\)) remains satisfiable along with satisfying \(F^q\). Hence, from every reachable state in the top game, the controller must be able to give one explicit trace which visits both \(p'_1\) and \(p'_2\) always eventually again. This prevents the controller from moving to a state in the top game where the plant’s assumptions are persistently violated.

A gracious strategy in the original obliging game is extracted from this Gedankenexperiment as follows. First, we consider the upper and the lower game in Fig. 3 separately. For the upper game, we know that a supervisor disabling events \(a\) and \(b\) in every state is winning w.r.t. \(L_\prec\) (see Example IV.1). Call this strategy \(h^1\). We can assume w.l.o.g. that \(h^1\) is memoryless, because, in any Rabin game, if there is a winning strategy, there is also a memoryless one. This strategy forces the plant to always remain in \(p_0\) and wins in the upper game by vacuously satisfying the implication. For the lower part, consider the blue transitions in Fig. 4, indicating an infinite trace from every state visiting both \(p_1\) and \(p_2\) infinitely often, fulfilling both \(S = F^p\) and \(W = F^q\). This path immediately defines a memoryless plant and a control player strategy which we denote \(g^1\) and \(h^1\), respectively.

Given \(h^1\), \(g^1\), and \(h^1\), we can combine them into a solution to the original synthesis problem over \(G(M)\) (and therefore \(M\)) in Fig. 2, by adding one extra bit of memory to the controller. That is, the resulting strategy will base its decision on the current state and an additional binary-valued variable \(m\) which tracks, whether the system executes a move contained in \(g^1\) (\(m = 1\)) or not (\(m = 0\)). If \(m = 1\), the controller executes the unique pattern chosen by \(h^1\) in the next state. Otherwise, it operates according to \(h^1\).

For the particular choices of strategies in this example we see that the only allowed event in \((p_0, \gamma)\) is part of \(g^1\) and, therefore, triggers \(h^1\). Hence, the actual closed loop allows the plant to move to \(p_2\) next. If it does so, \(h^1\) remains active as this move is again contained in \(g^1\) (see Fig. 4). If the plant decides to stay in \(p_0\), \(h^1\) becomes active again. Intuitively, the controller tracks whether the plant is trying to make progress towards fulfilling her liveness condition. If so, he is cooperating with her to achieve this goal.

### D. Algorithm

The reduction outlined in the previous section via Theorems III.2 and V.2 enables us to solve a given supervisory synthesis problem (Problem 1) over a plant model \((L, q, L_P, L_R, \Sigma)\) and a set of uncontrollable events \(\Sigma_{uc} \subseteq \Sigma\) through the following steps.

1. Construct a Street/Rabin automaton \((M, F^p, F^q)\) as in Proposition III.1.
2. Extend \(M\) into a game graph \(G(M)\) as in Definition IV.1.
3. Solve the obliging game \((G(M), F^p_{\prec,S}, F^q)\) via its reduction to standard \(\omega\)-regular games (see Theorem V.1).
4. If the obliging game has no solution, also Problem 1 has no solution (see Theorems III.2 and V.2).
5. If the obliging game allows for a control strategy \(h\), compute its induced nonambiguous strategy \(h^1\) as in (10).
6. Reduce \(h^1\) to a path-based supervisor via (11), which in turn defines the event-based supervisor \(f^1\) via (3).
7. Then, \(f^1\) solves Problem 1 (see Theorems III.2 and V.2).

The complexity of this algorithm can be derived from Theorem V.1 in the following way. Given a finite-state machine \(M\) with \(n\) states we get a game graph \(G(M)\) with \(n2^{2^\omega}\) states. Furthermore, given the Street and Rabin conditions \(F^p\) and \(F^q\) with \(l\) and \(k\) pairs, we get an obliging game having a strong Rabin condition with \(l + k\) pairs and a weak Street condition with \(l\) pairs. Finally, a parity game with \(n\) states and \(k\) colors can be solved in \(O(n^2\omega)\) time. Hence, our solution can be computed in time \(O((n2^{2^{\omega}}(l + k)^2(l + k))2^{O(\omega(l + k)^2)})\). If there is a supervisor, then there is a supervisor using 2\((l + k) \cdot 2^{O(l)}\) memory.

It should further be noted that checking if there is a path-based supervisor from a state is NP-complete [7]; this already holds for a trivial liveness assumption for the plant (i.e., \(L_P = \Sigma^*\)) as solving Rabin games is NP-complete [21]. While our algorithm is sound and complete, it is possible that there is a more direct symbolic algorithm on the state space of the two-person game that yields a more efficient implementation. Such an algorithm is given in [23] for the special case where \(F^p\) and \(F^q\) are each a generalized Büchi winning condition. We postpone the generalization of this algorithm to future work.

### References

[1] P. J. Ramadge and W. M. Wonham, “Supervisory control of a class of discrete event processes,” *SIAM J. Control Optim.*, vol. 25, pp. 206–237, 1987.
[2] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Berlin, Germany: Springer, 2008.
[3] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems* (Communications and Control Engineering Series). Berlin, Germany: Springer, 2019.
[4] J. G. Thistle and W. M. Wonham, “Control of \(\omega\)-automata, Church’s problem, and the emptiness problem for tree \(\omega\)-automata,” in *Proc. Int. Workshop Comput. Sci. Log.*, 1991, pp. 367–381.
[5] J. G. Thistle and W. M. Wonham, “Supervision of infinite behavior of discrete-event systems,” *SIAM J. Control Optim.*, vol. 32, no. 4, pp. 1088–1113, 1994.
[6] J. G. Thistle and W. M. Wonham, “Control of infinite behavior of finite automata,” *SIAM J. Control Optim.*, vol. 32, no. 4, pp. 1075–1097, 1994.
[7] J. G. Thistle, “On control of systems modelled as deterministic Rabin automata,” *Discrete Event Dyn. Syst.*, vol. 5, no. 4, pp. 357–381, 1995.
[8] A. Church, “Applications of recursive arithmetic to the problem of circuit synthesis,” *Summaries Summer Inst. Symbolic Log.*, vol. I, pp. 3–50, 1957.
[9] O. Maler, A. Pnueli, and J. Sifakis, “On the synthesis of discrete controllers for timed systems,” in Proc. Annu. Symp. Theor. Aspects Comput. Sci., 1995, pp. 229–242.

[10] J. Lygeros, C. Tomlin, and S. Sastry, “Controllers for reachability specifications for hybrid systems,” Automatica, vol. 35, no. 3, pp. 349–370, 1999.

[11] J. M. Davoren and A. Nerode, “Logics for hybrid systems,” Proc. IEEE, vol. 88, no. 7, pp. 985–1010, Jul. 2000.

[12] R. Ehlers, S. LaFortune, S. Tripakis, and M. Y. Vardi, “Supervisory control and reactive synthesis: A comparative introduction,” Discrete Event Dyn. Syst., vol. 27, no. 2, pp. 209–260, 2017.

[13] A. Schmuck, T. Moor, and K. W. Schmidt, “A reactive synthesis approach to supervisory control of terminating processes,” in Proc. 21st IFAC World Congr., 2020, pp. 2149–2156.

[14] Z. Ramezani, J. Krook, Z. Fei, M. Fabian, and K. Akesson, “Comparative case studies of reactive synthesis and supervisory control,” in Proc. 18th Eur. Control Conf., 2019, pp. 1752–1759.

[15] D. A. Ciolek, V. Braberman, N. D’Ippolito, S. Sardiña, and S. Uchitel, “Compositional supervisory control via reactive synthesis and automated planning,” IEEE Trans. Autom. Control, vol. 65, no. 8, pp. 3502–3516, Aug. 2019.

[16] K. Chatterjee, F. Horn, and C. Löding, “Obliging games,” in Proc. Int. Conf. Concurrency Theory, 2010, pp. 284–296.

[17] B. Alpern and F. B. Schneider, “Recognizing safety and liveness,” Distrib. Comput., vol. 2, no. 3, pp. 117–126, 1987.

[18] R. Majumdar and A.-K. Schmuck, “Supervisory controller synthesis for non-terminating processes is an obliging game,” 2020. [Online]. Available: https://arxiv.org/abs/2007.01773

[19] E. Emerson and C. Jutla, “Tree automata, mu-calculus and determinacy,” in Proc. 32nd Ann. Symp. Found. Comput. Sci., 1991, pp. 368–377.

[20] A. Pnueli and R. Rosner, “On the synthesis of a reactive module,” in Proc. 16th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang., 1989, pp. 179–190.

[21] E. A. Emerson and C. S. Jutla, “The complexity of tree automata and logics of programs,” in Proc. 29th Ann. Symp. Found. Comput. Sci., 1988, vol. 88, pp. 328–337.

[22] W. Thomas, “On the synthesis of strategies in infinite games,” in Proc. Ann. Symp. Theor. Aspects Comput. Sci., 1995, pp. 1–13.

[23] R. Majumdar, N. Piterman, and A.-K. Schmuck, “Environmentally-friendly GR(1) synthesis,” in Proc. Int. Conf. Tools Algorithms Construction Anal. Syst., 2019, pp. 229–246.