Supervisory Controller Synthesis for Non-terminating Processes is an Obliging Game

Rupak Majumdar and Anne-Kathrin Schmuck

arXiv:2007.01773v3 [cs.LO] 26 Aug 2021

***Abstract*—We present a new algorithm to solve the supervisory control problem over non-terminating processes modeled as ω-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 ω-regular reactive synthesis. Therefore, our reduction results in a symbolic algorithm based on manipulating sets of states using tools from reactive synthesis.**

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 text books [2], [3].

Already 30 years ago, Thistle and Wonham extended the scope of SCT to *non-terminating processes* [4], i.e., to the supervision of systems whose behavior can be modeled by regular languages over *infinite* words. Non-terminating processes naturally occur in models of infinitely executing reactive systems; ω-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 non-terminating processes and showed, in particular, an algorithm to synthesize supervisors for general ω-regular specifications under general ω-regular plant properties. This symbolic synthesis algorithm involves an intricate fixed point computation over the ω-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 our 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 arguable also influenced early works on using logical formalisms to control hybrid systems [9]–[11] which partially culminated into the now well established field of formal methods for hybrid control. However, despite this lasting connection between hybrid control and logic, the connection between the research fields of *discrete* supervisory control and reactive synthesis got mostly lost over time and is just about to be re-established. This paper continues these recent efforts [12]–[15] by showing a new clean algorithmic connection between SCT and recent enhancements of reactive synthesis, called *obliging games*.

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* ω-regular languages defined over this machine. The first language (let's 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 above implication vacuously by actively preventing the plant from satisfying the assumption. In SCT, such undesired solutions are ruled out by a *non-conflicting* requirement: any finite word compliant with the supervisor must be extendable to an infinite word that satisfies A. Hence, a non-conflicting supervisor always allows the plant to fulfill the assumption. The non-conflicting requirement is not a linear property [12], and cannot be "compiled away" in reactive synthesis.

The main contribution of this paper 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 non-conflicting 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

R. Majumdar and A.-K. Schmuck are with MPI-SWS, Kaiserslautern, Germany (e-mail: {rupak,akschmuck}@mpi-sws.org). Authors are ordered alphabetically.

non-conflicting strategies and (b) the (non-trivial, but well-understood) algorithmics of solving games.

Other Related Work: This paper continues recent efforts in establishing a formal connection between reactive synthesis and SCT for *terminating processes* [12]–[15] and *non-terminating processes* [17]. While [17] focusses on a language-theoretic connection, this paper establishes a connection between synthesis algorithms over automata realizations.

Within the SCT community, non-terminating processes have gained more attention in recent years, see e.g., [18]–[21]. However, in all these works, the plant itself does not posses non-trivial liveness properties, which allows to transform the resulting synthesis problem to the usual setting of reactive synthesis. Notable exceptions are, e.g., [22], [23], where synthesisis is restricted to *deterministic* Büchi automata models, capturing only a strict subclass of ω-regular properties.

Symbolic algorithms for GR(1) specifications satisfying a non-conflicting requirement were presented in [24]. Their algorithm has the advantage of a "direct" implementation using symbolic manipulation of sets of states. We leave as future work whether a similar direct algorithm can be designed for general obliging games.

II. Preliminaries

Formal Languages. Given a finite alphabet Σ, we write Σ^*, Σ^+, and Σ^ω for the sets of finite words, non-empty finite words, and infinite words over Σ, and write $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We call the subsets $L \subseteq \Sigma^*$ and $\mathcal{L} \subseteq \Sigma^\omega$ a $*$-language and an ω-language over Σ, 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 $w \in \Sigma^\infty$ is a $*$-language denoted by $\mathrm{pfx}(w) \subseteq \Sigma^*$. For $L \subseteq \Sigma^*$, we have $L \subseteq \mathrm{pfx}(L)$. A $*$-language L is *prefix-closed* if $L = \mathrm{pfx}(L)$. The *limit* $\lim(L)$ of a $*$-language L is the ω-langauge which contains all words $\alpha \in \Sigma^\omega$ which have infinitely many prefixes in L. We further define $\mathrm{clo}(\mathcal{L}) := \lim(\mathrm{pfx}(\mathcal{L}))$ as the *topological closure* of $\mathcal{L} \subseteq \Sigma^\omega$. An ω-language $\mathcal{L}$ is *topologically-closed* if $\mathcal{L} = \mathrm{clo}(\mathcal{L})$.

Finite State Machines. A *finite state machine* is a tuple $M = (X, \Sigma, \delta, x_0)$, with *state set* X, *alphabet* Σ, *initial state* $x_0 \in X$, and the partial *transition function* $\delta : X \times \Sigma \rightharpoonup 2^X$. For $x \in X$ and $\sigma \in \Sigma$, we write $\delta(x, \sigma)!$ to signify that $\delta(x, \sigma)$ is defined. We call M *deterministic* if $\delta(x, \sigma)!$ implies $|\delta(x, \sigma)| = 1$. We call M *non-blocking* if for all $x \in X$ there exists at least one $\sigma \in \Sigma$ s.t. $\delta(x, \sigma)!$.

A *path* of M is a finite or infinite sequence $\pi = x_0 x_1 \ldots$ s.t. for all $k \in \mathsf{Length}(\pi) - 1$ there exists some $\sigma_k \in \Sigma$ s.t. $x_{k+1} \in \delta(x_k, \sigma_k)$. If π is finite, we denote by $\mathsf{Last}(\pi) = x_n$ its last element. We collect all finite and infinite paths of the finite state machine M in the sets $P(M) \subseteq x_0 X^*$ and $\mathcal{P}(M) \subseteq x_0 X^\omega$, respectively. Given a string $s = \sigma_0 \sigma_1 \ldots \in \Sigma^\infty$ we say that a path π of M is *compliant* with s if $\mathsf{Length}(s) = \mathsf{Length}(\pi) - 1$ and for all $k \in \mathsf{Length}(\pi) - 1$ we have $x_{k+1} \in \delta(x_k, \sigma_k)$. We define by $\mathsf{Paths}_M(s)$ 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 $L(M) := \{s \in \Sigma^* \mid \mathsf{Paths}_M(s) \neq \emptyset\}$ and $\mathcal{L}(M) := \{s \in \Sigma^\omega \mid \mathsf{Paths}_M(s) \neq \emptyset\}$, respectively. If M is non-blocking, we have $\mathrm{pfx}(\mathcal{L}(M)) = L(M)$. If M is deterministic we have $|\mathsf{Paths}_M(s)| = 1$ for all $s \in L(M)$.

Finite-State Automata over Finite Words. Deterministic finite-state automata over finite words are typically called deterministic finite automata (DFA) and are defined by a deterministic finite state machine M equipped with a set of final states $F \subseteq X$. The DFA (M, F) accepts (or generates) the $*$-language $L(M, F)$ which contains all finite paths of M which are ending in F. A $*$-language L is called *regular* iff there exists a DFA (M, F) which accepts L, i.e., $L = L(M, F)$.

Finite-State Automata over Infinite Words. For a path π, define $\mathrm{Inf}(\pi) = \{x \in X \mid x_k = x \text{ for infinitely many } k \in \mathbb{N}\}$ to be the set of states visited infinitely often along π. Let $F \subseteq X$ be a subset of states. We say that an *infinite* string $s \in \Sigma^\omega$ satisfies the *Büchi acceptance condition* $\mathcal{F}^{\mathrm{B}} = \{F\}$ on M if there exists a path $\pi \in \mathsf{Paths}_M(s)$ such that $\mathrm{Inf}(\pi) \cap F \neq \emptyset$. Further, let $\mathcal{F} = \{\langle G_1, R_1 \rangle, \ldots, \langle G_m, R_m \rangle\}$ be a set, where each $G_i, R_i \subseteq X$, $i = 1, \ldots, m$, is a subset of states. We say that a string $s \in \Sigma^\omega$ satisfies the *Rabin acceptance condition* $\mathcal{F}^{\mathrm{R}} = \mathcal{F}$ on M if there exists a path $\pi \in \mathsf{Paths}(M, s)$ such that $\mathrm{Inf}(\pi) \cap G_i \neq \emptyset$ *and* $\mathrm{Inf}(\pi) \cap R_i = \emptyset$ for *some* $i \in [1; m]$. It satisfies the *Streett acceptance condition* $\mathcal{F}^{\mathrm{S}} = \mathcal{F}$ if $\mathrm{Inf}(\pi) \cap G_i = \emptyset$ *or* $\mathrm{Inf}(\pi) \cap R_i \neq \emptyset$ for *all* $i \in [1; m]$. Rabin and Streett conditions are *duals*, i.e., if π satisfies the Rabin condition $\mathcal{F}^{\mathrm{R}}$ it violates the Streett condition $\mathcal{F}^{\mathrm{S}} = \mathcal{F}^{\mathrm{R}}$.

We call a finite state machine equipped with a Büchi, Rabin or Streett acceptance condition a Büchi, Rabin or Streett automaton, respectively. We collect all infinite strings (resp. paths) satisfying the specified acceptance condition $\mathcal{F}$ over M, in the accepted language $\mathcal{L}(M, \mathcal{F}) \subseteq \Sigma^\omega$ (resp. in the set $\mathcal{P}(M, \mathcal{F}) \subseteq x_0 X^\omega$). An ω-language $\mathcal{L}$ is called *regular* iff it is accepted by a *non-deterministic* Büchi automaton. We remark that *deterministic* Rabin and *deterministic* Streett automata also accept precisely the set of ω-regular languages. However, this is not true for *deterministic* Büchi automata which are less expressive.

III. The Supervisor Synthesis Problem

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

A. Problem Statement

Let Σ be a finite alphabet of *events*. A *plant* is a tuple $(L_P, \mathcal{L}_P)$, where $L_P \subseteq \Sigma^*$ is a prefix-closed regular $*$-language and $\mathcal{L}_P \subseteq \Sigma^\omega$ s.t. $\mathrm{pfx}(\mathcal{L}_P) \subseteq L_P$ is a regular ω-language. If, in addition, $\mathrm{pfx}(\mathcal{L}_P) = L_P$, the plant is called *deadlock-free*. A *specification* is a tuple $(L_S, \mathcal{L}_S)$ where $\mathcal{L}_S \subseteq \Sigma^\omega$ is a regular ω-language and $L_S := \mathrm{pfx}(\mathcal{L}_S)$ is a prefix-closed regular $*$-language. That is, the specification (in contrast to the plant) is by definition deadlock-free. This convention is motivated by the fact that any closed-loop system should be deadlock free in order to operate correctly, independent of other properties that should be enforced.

Intuitively, the language-tuples $(L_P, \mathcal{L}_P)$ and $(L_S, \mathcal{L}_S)$ capture both *safety* and *liveness* properties. Here, $(L_P, \mathcal{L}_P)$ models the properties the *uncontrolled* plant exhibits. In contrast, $(L_S, \mathcal{L}_S)$ 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.

Remark 1. In language theory, an ω-language $\mathcal{L}$ is called a *safety language* if $\lim(\mathrm{pfx}(\mathcal{L})) = \mathcal{L}$ (that is, a finite prefix of a string determines containment in a language), and a *liveness language* if $\lim(\mathrm{pfx}(\mathcal{L})) = \Sigma^\omega$ (that is, finite prefixes do not matter for containment). By using this classification of languages we can interpret the language tuples $(L_P, \mathcal{L}_P)$ and $(L_S, \mathcal{L}_S)$ from above as follows. First, we see that both L_P and L_S capture the safety-part of the plant and the specification, respectively, by observing that $\lim(L_P)$ and $\lim(L_S)$ are indeed safety languages. For the liveness part, the correspondence is not as straight forward. However, as we know that any regular language can be written as the intersection of a safety and a liveness language [25], there exist (pure) liveness languages $\tilde{\mathcal{L}}_P$ and $\tilde{\mathcal{L}}_S$ s.t. $\mathcal{L}_P = \tilde{\mathcal{L}}_P \cap \lim(L_P)$ and $\mathcal{L}_S = \tilde{\mathcal{L}}_S \cap \lim(L_S) = \tilde{\mathcal{L}}_S \cap \lim(\mathrm{pfx}(\mathcal{L}_S))$. Hence, in the context of SCT, the ω-languages $\mathcal{L}_P$ and $\mathcal{L}_S$ capture both safety and liveness properties.

For any interesting instance of the SCT control problem over infinite strings, we require $\mathcal{L}_P \subsetneq \lim(L_P)$ and $\mathcal{L}_S \subsetneq \lim(L_S)$, that is, both the plant and the specification languages $\mathcal{L}_P$ and $\mathcal{L}_S$ do not only contain a safety property (i.e., $\lim(L_P)$ and $\lim(L_S)$) but also a non-trivial liveness property (captured by $\mathcal{L}_P \setminus \lim(L_P)$ and $\mathcal{L}_S \setminus \lim(L_S)$, respectively). We refer the reader to [23] for an accessible discussion of this topic.

Given the finite alphabet Σ, its subset $\Sigma_\mathrm{c} \subseteq \Sigma$ denotes all events the controller can prevent the plant from executing, while the set $\Sigma_\mathrm{uc} \subseteq \Sigma$ denotes events that cannot be prevented by the controller. We typically require that Σ_c and Σ_uc form a partition of Σ, i.e., $\Sigma = \Sigma_\mathrm{c} \dot{\cup} \Sigma_\mathrm{uc}$.

Further, a *control pattern* γ is a subset of Σ containing Σ_uc. We collect all control pattern in the set $\Gamma := \{ \gamma \subseteq \Sigma \mid \Sigma_\mathrm{uc} \subseteq \gamma \}$. Given this set, a *(string-based) supervisor* is defined as a map $f : \Sigma^* \to \Gamma$ 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 $t\sigma \in \mathrm{pfx}(s)$, it holds that $\sigma \in f(t)$. We write L_f for the set of all words consistent with f and define $\mathcal{L}_f := \lim(L_f)$.

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

Problem 1 (String-Based Supervisor Synthesis)**.** Given an alphabet $\Sigma = \Sigma_\mathrm{c} \dot{\cup} \Sigma_\mathrm{uc}$, a plant model $(L_P, \mathcal{L}_P)$, where $\mathcal{L}_P \subseteq \Sigma^\omega$ and $\mathrm{pfx}(\mathcal{L}_P) \subseteq L_P \subseteq \Sigma^*$ are regular languages, and a regular specification language[1] $\mathcal{L}_S \subseteq \Sigma^\omega$, synthesize, if possible, a *string-based supervisor* $f : \Sigma^* \to \Gamma$ s.t.

(i) the closed-loop satisfies the specification, i.e.,

$$\emptyset \subsetneq \mathcal{L}_f \cap \mathcal{L}_P \subseteq \mathcal{L}_S \tag{1a}$$

(ii) the plant and the supervisor are *non-conflicting*, i.e.,

$$L_f \cap L_P \subseteq \mathrm{pfx}(\mathcal{L}_f \cap \mathcal{L}_P), \tag{1b}$$

or determine that no such supervisor exists. A string-based supervisor f solves the synthesis problem over $((L_P, \mathcal{L}_P), \mathcal{L}_S)$ if it satisfies (1a) and (1b). ◁

The 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 $\mathcal{L}_P$. Then, by (1a), all such generated words must be contained in the specification $\mathcal{L}_S$.

B. A Special Case: Terminating Processes

Given that many readers might be more familiar with the supervisory controller synthesis problem for *terminating* processes, we first recall its special case of Problem 1 and a standard algorithmic solution before discussing automata realizations for solving Problem 1 for ω-regular input parameters.

Within the basic setting of supervisory controller synthesis for *terminating processes*, the languages $\mathcal{L}_P$ and $\mathcal{L}_S$ are non-prefix closed regular $*$-languages, rather than regular ω-languages. They are typically called the *marked* language and denoted by L_{mP} and L_{mS}, respectively. In this setting, it is well known that the tuples (L_P, L_{mP}) and (L_S, L_{mS}) can be represented by deterministic finite automata (DFA) denoted by (M_P, F_P) and (M_S, F_S), s.t. $L_P := L(M_P)$, $L_{mP} = L(M_P, F_P)$, $L_S := L(M_S)$, and $L_{mS} = L(M_S, F_S)$. I.e., L_P and L_S collect all *finite* strings generated by M_P and M_S, respectively, when starting form the initial state and ending in any other state $x \in X$. Similarly, L_{mP} and L_{mS} collect all *finite* strings generated by M_P and M_S, respectively, when starting form the initial state and ending in a *marked* state. As in the ω-language case, it is further assume that (M_S, F_S) is non-blocking, i.e., $L_S = \mathrm{pfx}(L_{mS})$. In addition, one typically requires that L_{mS} is closed w.r.t. L_{mP}.

In the standard version of the supervisory control problem over terminating processes (see e.g. [26], p.184) one usually refers to the intersections $L_f \cap L_P$ and $L_f \cap L_{mP}$ as the unmarked and marked language of the closed-loop (i.e., the plant under supervision), usually denoted by $L(f/P)$ and $L_m(f/P)$. Then the standard supervisory control problem[2] asks for a supervisor $f : \Sigma^* \to \Gamma$, s.t. the closed-loop is non-blocking, i.e., $L(f/P) = \mathrm{pfx}(L_m(f/P))$ and $\mathrm{pfx}(L_m(f/P))$ is the maximal language (with the above properties) contained in the marked specification language L_{mS}.

Matching these two requirements to Problem 1 we see that (1b) corresponds to the non-blocking requirement, while (1a) requires containment in the marked specification language. The fact that the classical supervisory synthesis problem asks for a maximal solution stems from the fact that such a maximal solution does uniquely exist for *terminating* processes. It

[1] As $L_S := \mathrm{pfx}(\mathcal{L}_S)$, the language L_S is uniquely determined by $\mathcal{L}_S$ and therefore omitted from the problem description.

[2] Note that the usual controllablity requirement is hidden in the definition of f to always enable uncontrollable events.

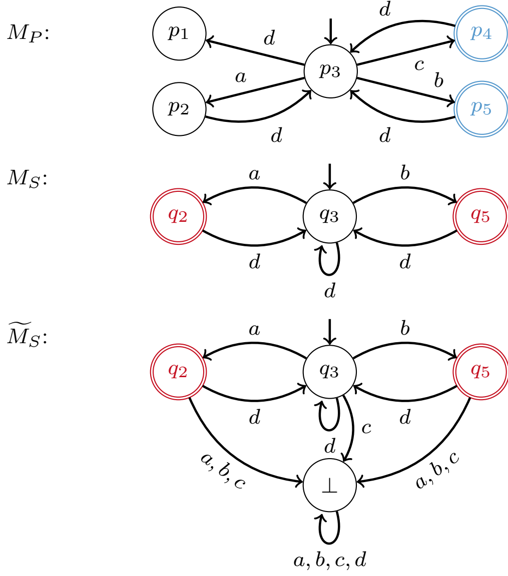

Fig. 1. Example automata for the construction of synthesis automta. For terminating processes, we interpret them as DFA (M_P, F_P), (M_S, F_S) and $(\widetilde{M}_S, F_S)$ and for non-terminating processes we interpret them as Büchi automata $(M_P, \mathcal{F}_P)$, $(M_S, \mathcal{F}_S)$ and $(\widetilde{M}_S, \mathcal{F}_S)$ with $\mathcal{F}_i = \{F_i\}$, $i \in \{P, S\}$. States in the set F_i are indicated by double circles.

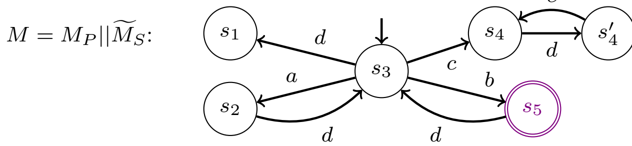

Fig. 2. Synthesis automaton (M, F) for the supervisory controller synthesis problem over *terminating processes* depicted in Fig. 1. States in F are indicated by violet douple circled states.

is worth noting, that this is in general not true for *non-terminating* processes, and hence omitted from Problem 1.

One standard algorithmic solution to the outlined supervisor synthesis problem (see [26], p.186) combines both DFA realizations (M_P, F_P) and (M_S, F_S) into a single DFA (M, F). This process is sometimes called "plantification of the specification". This construction essentially extends the DFA (M_S, F_S) into a complete DFA $(\widetilde{M}_S, F_S)$ whose unmarked language is unrestricted, i.e., $L(\widetilde{M}_S) = \Sigma^*$ and then takes a normal automata product of (M_P, F_P) and $(\widetilde{M}_S, F_S)$ to obtain the synthesis automaton (M, F) where a state (q, p) is marked (i.e., contained in F) if $q \in F_P$ and $p \in F_S$. Intuitively, solving Problem 1 reduces to strategically disabling controllable transition in M s.t. states in F always remain reachable, which ensures safety, non-blockingness and controllability. It is proven in (see [26], p.186) that this construction is indeed correct and results in the desired supervisor.

As an example, consider the DFA's (M_P, F_P) and (M_S, F_S) depicted in Fig. 1 (top) and (middle) and the completed DFA $(\widetilde{M}_S, F_S)$ in Fig. 1 (bottom). Their product results in the synthesis automaton (M, F) depicted in Fig. 2. Assuming that all events are controllable, the resulting supervisor would disable events d and c in s_3.

C. Automata Representations for Supervisor Synthesis

The intuition behind the automata realization for solving Problem 1 directly carries over from terminating processes (as discussed in Sec. III-B) to non-terminating processes. If $\mathcal{L}_P$ and $\mathcal{L}_S$ are indeed non-(topologically) closed regular ω-languages, the language tuples $(L_P, \mathcal{L}_P)$ and $(L_S, \mathcal{L}_S)$ are realized by Büchi, Rabin or Streett automata, instead of DFA's. I.e., as all involved languages are regular, there exist automata $(M_P, \mathcal{F}_P)$ $(M_S, \mathcal{F}_S)$ s.t. L_P and L_S collect all *finite* strings compliant with M_P and M_S, and $\mathcal{L}_P$ and $\mathcal{L}_S$ collect all *infinite* strings that are compliant with M_P and M_S, respectively, and, in addition, satisfy the given acceptance condition $\mathcal{F}_P$ and $\mathcal{F}_S$ over M_P and M_S, respectively. As any ω-regular language is accepted (or generated) by a *deterministic* Rabin or Streett automaton, we can, without loss of generality, follow [7] and assume that $(M_P, \mathcal{F}_P)$ is a deterministic Streett automaton, while $(M_S, \mathcal{F}_S)$ is a deterministic Rabin automaton.

Similar to the $*$-language case, both ω-automata can be combined to a single machine M while using the premisse of Problem 1 that $(M_S, \mathcal{F}_S)$ is non-blocking, i.e., $L(M_S) = \mathrm{pfx}(\mathcal{L}(M_S, \mathcal{F}_S))$. The major difference here is that ω-regular acceptance conditions do not require "synchonization", that is, plant and specification markings need not be reached *at the same time*. An infinite path over M can fulfill the two different acceptance conditions $\mathcal{F}_P$ and $\mathcal{F}_S$ (for example visiting a state in $\mathcal{F}_P$ *and* a state in $\mathcal{F}_S$ infinitely often in the case of Büchi conditions) without requiring that both conditions are met *at the same time*. This is the reason why SCT for *non-terminating processes* does not require that the specification is closed w.r.t. the plant language. Further, this implies that the product of the ω-automata $(M_P, \mathcal{F}_P)$ and $(M_S, \mathcal{F}_S)$ results in a single automaton M equipped with *two* sets of marked states $\mathcal{F}'_P$ and $\mathcal{F}'_S$ over M. This is formalized in the following definition.

Definition III.1. Let $M_P = (X_P, \Sigma, \delta_P, x_{P,0})$ and $M_S = (X_S, \Sigma, \delta_S, x_{S,0})$ be two deterministic state machines over the same alphabet Σ with $\mathcal{F}_P = \{\langle G_{P,1}, R_{P,1}\rangle, \ldots, \langle G_{mP}, R_{mP}\rangle\}$ and $\mathcal{F}_S = \{\langle G_{S,1}, R_{S,1}\rangle, \ldots, \langle G_{S,n}, R_{S,n}\rangle\}$ being a Streett and a Rabin acceptance condition over X_P and X_S, respectively. Further, let $\widetilde{M}_S = (X_S \cup \{\perp\}, \Sigma, \widetilde{\delta}_S, x_{S,0})$ be the completion of M_S, that is, for all $x \in X_S$ and $\sigma \in \Sigma$ holds that $\perp = \widetilde{\delta}(x, \sigma)$ iff $\delta(x, \sigma)$ is undefined, and $\perp = \widetilde{\delta}(\perp, \sigma)$ for all $\sigma \in \Sigma$.

Then we define the extended product of $(M_P, \mathcal{F}_P)$ and $(M_S, \mathcal{F}_S)$ as the tuple $(M, \mathcal{F}'_P, \mathcal{F}'_S)$ with

- $M = (X, \Sigma, \delta, x_0)$ s.t. $X = X_P \times \widetilde{X}_S$, $q_0 = (x_{P,0}, x_{S,0})$, and $(q', p') = \delta((q, p), \sigma)$ iff $q' = \delta_P(q, \sigma)$ and $p' = \widetilde{\delta}_S(p, \sigma)$,
- $\mathcal{F}'_P = \{\langle G'_{P,1}, R'_{P,1}\rangle, \ldots, \langle G'_{mP}, R'_{mP}\rangle\}$ s.t. $G'_{P,i} = \{(p, q) \in X \mid p \in G_{P,i}\}$ and $R'_{P,i} = \{(p, q) \in X \mid p \in R_{P,i}\}$, and
- $\mathcal{F}'_S = \{\langle G'_{S,1}, R'_{S,1}\rangle, \ldots, \langle G'_{mS}, R'_{mS}\rangle\}$ s.t. $G'_{S,i} = \{(p, q) \in X \mid q \in G_{S,i}\}$ and $R'_{S,i} = \{(p, q) \in X \mid q \in R_{S,i}\}$.

Due to the usual properties of automata completion and product, we have the following observations.

Lemma III.1. *Given the premisses of Def. III.1, it holds that (a) $L(M_P) = L(M)$, (b) $\mathcal{L}(M_P, \mathcal{F}_P) = \mathcal{L}(M, \mathcal{F}'_P)$, (c) $\mathcal{L}(M_S, \mathcal{F}_S) \cap \mathcal{L}(M) = \mathcal{L}(M, \mathcal{F}'_S)$ and (d) M is deterministic.*

Proof. First, recall that completing the finite state machine M_S into $\widetilde{M}_S$ implies $L(\widetilde{M}) = \Sigma^*$ and $\mathcal{L}(\widetilde{M}) = \Sigma^\omega$. Further, observe that the construction of M is the usual product of M_P and $\widetilde{M}_S$, implying that M is deterministic (i.e., (d) holds), $L(M) = L(M_P) \cap L(\widetilde{M}_S) = L(M_P) \cap \Sigma^* = L(M_P)$ and $\mathcal{L}(M) = \mathcal{L}(M_P) \cap \mathcal{L}(\widetilde{M}_S) = \mathcal{L}(M_P) \cap \Sigma^\omega = \mathcal{L}(M_P)$. With this (a) directly holds. Further, $\mathcal{L}(M) = \mathcal{L}(M_P)$ and the construction of $\mathcal{F}'_P$ from $\mathcal{F}_P$ implies that a string $\omega \in \mathcal{L}(M)$ fulfills $\mathcal{F}'_P$ iff it fulfills $\mathcal{F}_P$, hence (b) holds. Similarly, the construction of $\mathcal{F}'_S$ from $\mathcal{F}_S$ implies that a string $\omega \in \mathcal{L}(M)$ fulfills $\mathcal{F}'_S$ iff it fulfills $\mathcal{F}_S$, implying (c). ∎

In order to use the automaton $(M, \mathcal{F}'_P, \mathcal{F}'_S)$ to solve Problem 1 we additionally need that distinct transitions in M carry distinct labels, i.e. for any $\sigma, \sigma' \in \Sigma$ and $x \in X$ we have that $\delta(x, \sigma) = \delta(x, \sigma')$ implies $\sigma = \sigma'$. This can be enforced by the following construction.

Definition III.2. Let $(M, \mathcal{F}_P, \mathcal{F}_S)$ be a deterministic extended product automaton. Then we define its *distinct transition version* as the tuple $(M', \mathcal{F}'_P, \mathcal{F}'_S)$ s.t.
(a) $M' := (X \times \Sigma,\ \Sigma,\ \delta',\ x_0)$ with $(x', \sigma) = \delta'(x_0, \sigma)$ iff $x' = \delta(x_0, \sigma)$ and for all $\overline{\sigma} \in \Sigma$ holds that $(x', \sigma) = \delta'((x, \overline{\sigma}), \sigma)$ iff $x' = \delta(x, \sigma)$,
(b) $\mathcal{F}'_P := \{(x, \sigma) \in X' \mid x \in \mathcal{F}_P\} \cup (\{x_0\} \cap \mathcal{F}_P)$, and
(c) $\mathcal{F}'_S := \{(x, \sigma) \in X' \mid x \in \mathcal{F}_S\} \cup (\{x_0\} \cap \mathcal{F}_S)$.

The above construction is trivially language preserving, i.e., we have the following lemma.

Lemma III.2. *Given the premises of Def. III.2, it holds that (a) $L(M) = L(M')$, (b) $\mathcal{L}(M, \mathcal{F}_P) = \mathcal{L}(M', \mathcal{F}'_P)$, (c) $\mathcal{L}(M, \mathcal{F}_S) = \mathcal{L}(M', \mathcal{F}'_S)$, (d) M' is deterministic and (e) distinct transitions in M' carry distinct labels.*

Proof. We first note that determinicity of M' follows from the fact that M is deterministic and we are only further splitting transitions, and not merging them. In addition M' has a single initial state. As both M and M' are deterministic, we further see that for every finite string $\alpha \in \Sigma^*$ a unique state is reached s.t. $\delta(x_0, \alpha) = x$ iff $\delta'(x_0, \alpha) = (x, \mathsf{last}(\alpha))$. This immediately shows (a) and implies from the definition of $\mathcal{F}'_P$ and $\mathcal{F}'_S$ from $\mathcal{F}_P$ and $\mathcal{F}_S$ that (b) and (c) also hold. (e) immediately follows from the construction. ∎

Summarizing the discussion above and recalling that any regular ω-language is realizabel by a deterministic Streett or Rabin automaton, we see that every input $((L_P, \mathcal{L}_P), \mathcal{L}_S)$ to Problem 1 can be realized by a so called *Streett/Rabin supervisor synthesis automaton* $(M, \mathcal{F}^{\mathsf{S}}_P, \mathcal{F}^{\mathsf{R}}_S)$. This is summarized in the following proposition.

Proposition III.3. *Let $L_P \subseteq \Sigma^*$, $\mathcal{L}_P, \mathcal{L}_S \subseteq \Sigma^\omega$ be regular languages and $((L_P, \mathcal{L}_P), \mathcal{L}_S)$ an input to Problem 1. Then there exists a finite state machine $M = (X,\ \Sigma,\ \delta,\ x_0)$, a Street condition $\mathcal{F}^{\mathsf{S}}_P$ over M and a Rabin condition $\mathcal{F}^{\mathsf{R}}_S$ over M, s.t. (a) $L_P = L(M)$, (b) $\mathcal{L}_P = \mathcal{L}(M, \mathcal{F}^{\mathsf{S}}_P)$, (c) $\mathcal{L}_S \cap \mathcal{L}(M) = \mathcal{L}(M, \mathcal{F}^{\mathsf{R}}_S)$, (d) M is deterministic, and (e) distinct transitions in M carry distinct labels, i.e. for any $\sigma, \sigma' \in \Sigma$ and $x \in X$ we have that $\delta(x, \sigma) = \delta(x, \sigma')$ implies $\sigma = \sigma'$.*

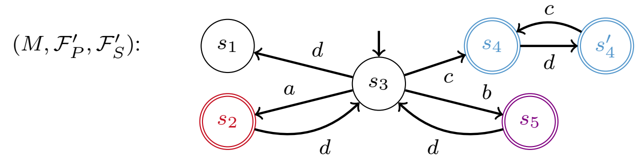

Fig. 3. Streett/Rabin supervisor synthesis automaton $(M, \mathcal{F}'_P, \mathcal{F}'_S)$ resulting from the Büchi automata $(M_P, \mathcal{F}_P)$, $(M_S, \mathcal{F}_S)$ and $(\widetilde{M}_S, \mathcal{F}_S)$ depicted in Fig. 1 with $\mathcal{F}'_P = \{s_4, s'_4, s_5\}$ (blue,violet) and $\mathcal{F}'_S = \{s_2, s_5\}$ (red,violet).

Proof. Let $((L_P, \mathcal{L}_P), \mathcal{L}_S)$ be an arbitrary regular input to Problem 1 over the alphabet Σ. As all languages are regular, there exists a *deterministic* Streett automaton $(M_P, \mathcal{F}_P)$ and a *deterministic* Rabin automaton $(M_S, \mathcal{F}_S)$ s.t. $L_P = L(M_P)$, $\mathcal{L}_P = \mathcal{L}(M_P, \mathcal{F}_P)$, $L_S := \mathrm{pfx}(\mathcal{L}_S) = L(M_S)$, $\mathcal{L}_S = \mathcal{L}(M_S, \mathcal{F}_S)$. Now first applying Def. III.1 to $(M_P, \mathcal{F}_P)$ and $(M_S, \mathcal{F}_S)$ yields the extended product automaton $(M^\times, \mathcal{F}^\times_P, \mathcal{F}^\times_S)$. Applying Def. III.2 to $(M^\times, \mathcal{F}^\times_P, \mathcal{F}^\times_S)$ yields its distrinct transition version $(M, \mathcal{F}^{\mathsf{S}}_P, \mathcal{F}^{\mathsf{R}}_S)$. With this, conditions (a)-(e) in Prop. III.3 immediately follow from Lem. III.1 and Lem. III.2. ∎

Definition III.3. If conditions (a)-(e) in Prop. III.3 are fulfilled, we call the tuple $(M, \mathcal{F}^{\mathsf{S}}_P, \mathcal{F}^{\mathsf{R}}_S)$ the *Streett/Rabin supervisor synthesis automaton* realizing $((L_P, \mathcal{L}_P), \mathcal{L}_S)$.

To illustrate the similarities and differences of the Streett/Rabin supervisor synthesis automaton with the synthesis automaton for terminating processes, we revisit the previous example and interpret the automata in Fig. 1 as ω-automata. As a deterministic Büchi automaton is a special case of both a Streett and a Rabin automaton[3], we simply interpret $(M_P, \mathcal{F}_P)$, $(M_S, \mathcal{F}_S)$ and $(\widetilde{M}_S, \mathcal{F}_S)$ as Büchi automata with $\mathcal{F}_i = \{F_i\},\ i \in \{P, S\}$. Now following the construction in Def. III.1 we obtain the Streett/Rabin supervisor synthesis automaton depicted in Fig. 3 which already has distinct transitions. We see that the Streett/Rabin supervisor synthesis automaton in Fig. 3 has both deadlocks (see state s_1) and livelocks (the loop between s_4 and s'_4 which does not allow to reach a specification marking infinitely often) which need to be prevented by the supervisor to imply safety of the closed loop. To additionally fulfill the liveness constrains imposed by the specification, the supervisor needs to make sure that every infinite trace of the controlled system visits red/violet states infinitely often if it visits blue/violet states infinitely often. For this example this trivially holds for every safe path. We will see in Sec. III-E that this is usually not the case.

D. Path-Based Supervisor Synthesis

We now formalize the outlined connection between effective algorithms to solve Problem 1 via a Streett/Rabin supervisor synthesis automaton $(M, \mathcal{F}^{\mathsf{S}}_P, \mathcal{F}^{\mathsf{R}}_S)$ realizing $((L_P, \mathcal{L}_P), \mathcal{L}_S)$,

[3]Let $(M, \{F\})$ be a Büchi automaton. Then this automaton is equivalent to the Rabin automaton $(M, \langle(F, \emptyset)\rangle)$ and the Streett automaton $(M, \langle(X, F)\rangle)$.

and Problem 1. This is done via a re-formulation of Problem 1 in terms of $(M,\mathcal{F}^{\mathsf{S}}_P,\mathcal{F}^{\mathsf{R}}_S)$ and so called *path-based* supervisors which is proven to be equivalent to Problem 1.

To this end, we define a *path-based supervisor* as a map $\check{f}: P(M)\rightarrow\Gamma$. A path π over M is called *consistent* with $\check{f}$ if for all $x,x'\in X$ and $\nu\in X^*$ s.t. $\nu xx'\in \mathrm{pfx}(\pi)$, there exists an event $\sigma\in\check{f}(\nu x)$ s.t. $x'=\delta(x,\sigma)$. Let $P(M,\check{f})$ be the set of all paths of M consistent with $\check{f}$ and define $\mathcal{P}(M,\check{f}):=\lim(P(M,\check{f}))$. We can now re-state Problem 1 into the following path-based supervisor synthesis problem.

Problem 2 (Path-based Supervision)**.** Given a Streett/Rabin supervisor synthesis automaton $(M,\mathcal{F}^{\mathsf{S}}_P,\mathcal{F}^{\mathsf{R}}_S)$, synthesize, if possible, a path-based supervisor $\check{f}: P(M)\rightarrow\Gamma$ s.t.

$$\emptyset\subsetneq\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)\subseteq\mathcal{P}(M,\mathcal{F}^{\mathsf{R}}_S),\ \text{and} \tag{2a}$$

$$P(M,\check{f})\subseteq \mathrm{pfx}\,(\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)), \tag{2b}$$

or determine that no such supervisor exists. A path-based supervisor $\check{f}$ solves the synthesis problem over $(M,\mathcal{F}^{\mathsf{S}}_P,\mathcal{F}^{\mathsf{R}}_S)$ 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 $\pi=\mathsf{Paths}_M(s)$ over M. Further, as transition labels are unique in M (see Prop. III.3 (e)), there is also a unique word s associated with a path π over M. With these observations, we can show that Problem 1 and 2 are indeed equivalent, as summarized by Thm. III.4.

Theorem III.4. *Let $L_P\subseteq\Sigma^*$, $\mathcal{L}_P,\mathcal{L}_S\subseteq\Sigma^\omega$ be regular languages. Let $(M,\mathcal{F}^{\mathsf{S}}_P,\mathcal{F}^{\mathsf{R}}_S)$ be a realizing Streett/Rabin supervisor synthesis automaton for the input $((L_P,\mathcal{L}_P),\mathcal{L}_S)$ to Problem 1. Further, let $f:\Sigma^*\rightarrow\Gamma$ and $\check{f}:P(M)\rightarrow\Gamma$ be a string- and a path-based supervisor, respectively, s.t.*

$$\forall s\in L(M)\ .\ f(s)=\check{f}(\mathsf{Paths}_M(s)). \tag{3}$$

Then f solves the synthesis problem over $((L_P,\mathcal{L}_P),\mathcal{L}_S)$ iff $\check{f}$ solves the synthesis problem over $(M,\mathcal{F}^{\mathsf{S}}_P,\mathcal{F}^{\mathsf{R}}_S)$.

Proof. We show both directions separately.
▶ "⇒": Fix f s.t. (1) holds, and $\check{f}$ s.t. (3) holds.
▷ We first show that $\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)\neq\emptyset$. To this end, recall that (1a) holds, i.e., $\mathcal{L}_f\cap\mathcal{L}_P\neq\emptyset$. This implies that there exists $s\in\mathcal{L}_P$ s.t. for all $\sigma\in\Sigma$ and $t\sigma\in\mathrm{pfx}\,(s)$ holds that $\sigma\in f(t)$. As M is deterministic, we have $|\mathsf{Paths}_M(s)|=1$ and define $\pi:=\mathsf{Paths}_M(s)$. With $\mathcal{L}_P=\mathcal{L}(M,\mathcal{F}^{\mathsf{S}}_P)$ we have $\pi\in\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)$. As M is deterministic, we have $\mathsf{Paths}_M(t\sigma)\in\mathrm{pfx}\,(\pi)\in P(M)$ for all $t\sigma\in\mathrm{pfx}\,(s)$. This implies $\pi\in\mathcal{P}(M,\check{f})$ from (3), i.e., $\pi\in\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)\neq\emptyset$.
▷ Show (2a): Now we pick any $\pi\in\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)$ and show that $\pi\in\mathcal{P}(M,\mathcal{F}^{\mathsf{R}}_S)$. First, as M is deterministic and has distinct transition labels, we have $|\mathsf{Paths}^{-1}_M(\pi)|=1$ and define $s:=\mathsf{Paths}^{-1}_M(\pi)$. As $\pi\in\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)$ we have $s\in\mathcal{L}(M,\mathcal{F}^{\mathsf{S}}_P)$. As $\mathcal{L}_P=\mathcal{L}(M,\mathcal{F}^{\mathsf{S}}_P)$ (from Prop. III.3 (b)) we have $s\in\mathcal{L}_P$. Further, as $\pi\in\mathcal{P}(M,\check{f})$ we know that for any $x,x'\in X$ and $\nu\in X^*$ s.t. $\nu xx'\in\mathrm{pfx}\,(\pi)$, there exists an event $\sigma\in\check{f}(\nu x)$ s.t. $x'=\delta(x,\sigma)$. Now it follows that indeed $s'=\mathsf{Paths}^{-1}_M(\nu x)$ is uniquely defined and $s'\sigma\in\mathrm{pfx}\,(s)$ by definition. Now it follows from (3) that $\sigma\in f(s')$ which implies $s\in\mathcal{L}_f$. I.e., we have $s\in\mathcal{L}_P\cap\mathcal{L}_f$. As (1a) holds we have $s\in\mathcal{L}_P$ and $s\in\mathcal{L}_S$. Now recall from Prop. III.3 (a) that $\mathcal{L}_P\subseteq\mathcal{L}(M)$ and therefore $s\in\mathcal{L}(M)\cap\mathcal{L}_S$. Now it follows from Prop. III.3 (c) that $\mathcal{L}_S\cap\mathcal{L}(M)=\mathcal{L}(M,\mathcal{F}^{\mathsf{R}}_S)$. This implies $\pi\in\mathcal{P}(M,\mathcal{F}^{\mathsf{R}}_S)$.
▷ Show (2b): Pick $\nu\in P(M,\check{f})$ and observe that this implies $\nu\in P(M)$. As $L_P=L(M)$ (from Prop. III.3 (a)) this implies that $t=\mathsf{Paths}^{-1}_M(\nu)\in L_P$. Further, it follows from the same reasoning as before that $t\in L_f$. As (1b) holds, this implies $t\in\mathrm{pfx}\,(\mathcal{L}_f\cap\mathcal{L}_P)$ and thereby $\nu\in\mathrm{pfx}\,(\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P))$.
▶ "⇐": Fix $\check{f}$ s.t. (2) holds, and f s.t. (3) holds.
▷ We first show that $\mathcal{L}_f\cap\mathcal{L}_P\neq\emptyset$. To this end, recall that (2a) holds, i.e., $\emptyset\subsetneq\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)$. Hence, there exists a path $\pi\in\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)$ s.t. also $\pi\in\mathcal{P}(M,\check{f})$. Now it follows form the same reasoning as in the proof of (2a) (in "⇒" above) that for $s:=\mathsf{Paths}^{-1}_M(\pi)$ holds $s\in\mathcal{L}_P\cap\mathcal{L}_f\neq\emptyset$.
▷ Show (1a): Now pick any $s\in\mathcal{L}_P\cap\mathcal{L}_f$ and show $s\in\mathcal{L}_S$. It again follows from the same reasoning as before (see first item in "⇒" above) that for $\pi:=\mathsf{Paths}_M(s)$ holds $\pi\in\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P)$. As (2a) holds, this implies $\pi\in\mathcal{P}(M,\mathcal{F}^{\mathsf{R}}_S)$ and therefore $s\in\mathcal{L}(M,\mathcal{F}^{\mathsf{R}}_S)$. Then it follows from condition (c) in Prop. III.3 that $s\in\mathcal{L}_S$.
▷ Show (1b): Pick $t\in L_f\cap L_P$ and define $\nu=\mathsf{Paths}^{-1}(t)$. Then it follows from the same reasoning as before that $t\in L_f$ implies $\nu\in P(M,\check{f})$. As (2b) holds, this implies $\nu\in\mathrm{pfx}\,(\mathcal{P}(M,\check{f})\cap\mathcal{P}(M,\mathcal{F}^{\mathsf{S}}_P))$ and thereby $t\in\mathrm{pfx}\,(\mathcal{L}_f\cap\mathcal{L}_P)$. ■

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 equipped with two acceptance conditions $\mathcal{F}_P$ and $\mathcal{F}_S$ where $\mathcal{F}_S$ is a Rabin condition. However, $\mathcal{F}_P$ is chosen to be trivial in [4] (i.e., $\mathcal{L}_P=\Sigma^\omega$), a deterministic Büchi condition in [5], [6], and a deterministic Streett condition in [7]. I.e., the algorithm in [7] solves Problem 2 but does not allow for a direct symbolic implementation. In the remaining sections of this paper, we show an alternative way to solve Problem 2 which establishes a new connection between the fields of supervisory control and reactive synthesis. This allows to utilize symbolic algorithms from reactive synthesis to solve Problem 2.

Remark 2. We remark that Thm. III.4 similarly holds for the $*$-language case. Here, the DFA (M,F) captures the synthesis problem for the $*$-language tuples $(L_P,\mathcal{L}_P)$ and $(L_S,\mathcal{L}_S)$ as discussed in Sec. III-C. In the classical supervisor synthesis procedure (see e.g. [2]), this synthesis automaton (M,F) is manipulated in various steps to compute a supervisor f solving Problem 1. As this synthesis procedure takes an automaton as input and computes a path-based supervisor as an output, it indeed solves Problem 2 (with input (M,F)). Hence, this algorithm only provides a solution to Problem 1 if Problem 1 and Problem 2 are indeed equivalent for every regular input, which is known to be true (see e.g., [2], [13] for details).

We want to point out that solving Problem 2 with ω-regular parameters is much more difficult then solving the (classical) $*$-language version. This is due to the fact that the ω-regular winning conditions of the plant and the specification (captured

by $\mathcal{F}_P^{\mathsf{S}}$ and $\mathcal{F}_S^{\mathsf{R}}$, respectively), do not need to be fulfilled simultaneously.

E. Another Simple Example

Consider the finite state machine M depicted in Fig. 4 for a path-based supervisor synthesis problem. Here, the alphabet is $\Sigma = \{a, b, c\}$, partitioned in $\Sigma_{\mathrm{c}} = \{a, b\}$ (indicated by a tick on the corresponding edges in Fig. 4) and $\Sigma_{\mathrm{uc}} = \{c\}$. 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 modelled by a Büchi acceptance condition $\mathcal{F}_P^{\mathsf{B}} = \{p_2\}$, and indicated by the light-blue double circle around state p_2 in Fig. 4. The Büchi condition $\mathcal{F}_P^{\mathsf{B}}$ can be equivalently formulated as the Streett condition $\mathcal{F}_P^{\mathsf{S}} = \{\langle\{p_0, p_1, p_2\}, \{p_2\}\rangle\}$.

The specification requires that the *controlled* plant should visit state p_1 always again (liveness) and does not dead-lock (safety). This can be modelled by a Büchi condition with $\mathcal{F}_S^{\mathsf{B}} = \{p_1\}$, indicated by the red double circle around p_1 in Fig. 4. Again, we can equivalently represent $\mathcal{F}_S^{\mathsf{B}}$ as the Rabin condition $\mathcal{F}_S^{\mathsf{R}} = \{\langle\{p_1\}, \emptyset\rangle\}$. 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 problem 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 b in state p_2.

Note that this solution is not unique: for each $n \geq 0$, a supervisor could map paths ending in p_0 and p_1 as before, but map paths ending in p_2 to $\{a, b, c\}$ if the number of visits to p_2 is less than n and to $\{a, c\}$ otherwise. In this case, the supervisor would allow the self loop on b in p_2 to be taken n number of times. A deterministic supervisor must however decide on a fixed n after which b is disabled in p_2, as otherwise the specification might not be fulfilled on a path fulfilling the plants' liveness assumption.

Remark 3. The above example demonstrates the well-known fact that for ω-languages, there may not exist a *maximally permissive* supervisor, i.e., a supervisor f solving Problem 1 s.t. $\mathcal{L}_{f'} \cap \mathcal{L}_P \subseteq \mathcal{L}_f \cap \mathcal{L}_P$ for all other supervisors f' solving Problem 1. In the above example, the maximal permissive supervisor would need to "eventually" disable b which cannot be modeled by a supervisor mapping from *finite* past strings to control patters. The induced language $\mathcal{L}_f$ of f is always topologically closed, disallowing the introduction of new liveness properties. This situation is in contrast to supervisor synthesis for $*$-languages where maximally permissive solutions to Problem 1 always exist.

IV. From Supervisor Synthesis to Games

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

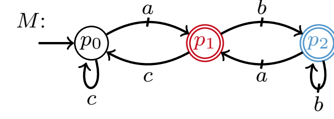

Fig. 4. Finite state machine M for example in Sec. III-E. Plant and specification markings indicated in blue (p_2) and red (p_1), respectively. Controllable events $\{a, b\}$ indicated by a ticked transition. Event c is uncontrollable.

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 $h : q_{init}(Q^1Q^0)^* \to Q^1$. The sequence $\rho \in Q^\infty$ is called a play over G if $\rho(0) = q_{init}$ and for all $k \in \mathsf{Length}(\rho) - 1$, we have $\rho(k+1) \in \delta^0(\rho(k))$ if $\mathsf{Last}(\rho) \in Q^0$ and $\rho(k+1) \in \delta^1(\rho(k))$ otherwise. The play ρ is *compliant* with h if additionally $h(\rho|_{[0,k]}) = \rho(k+1)$ if $\mathsf{Last}(\rho) \in Q^0$. 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 ω-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 $\mathcal{F}$ as defined in Sec. II over subsets of Q and say that a play ρ is winning w.r.t. $\mathcal{F}$ if ρ satisfies $\mathcal{F}$ on G. In addition, we also consider the *parity* accepting condition [27]. For the parity condition with k parities, we assume there is a coloring function $\Omega : Q \to \{0, \ldots, k-1\}$. A play ρ 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 $\mathcal{F}$ a Büchi, Rabin, Streett, or parity game, respectively, and denote it by the tuple $(G, \mathcal{F})$. The set of all winning plays over G w.r.t. $\mathcal{F}$ is denoted $\mathcal{P}(G, \mathcal{F})$. A strategy h is *winning* in a game $(G, \mathcal{F})$, if $\mathcal{P}(G, h) \subseteq \mathcal{P}(G, \mathcal{F})$. 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 [27]–[30].

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 non-determinism by choosing a transition allowed by γ. We formalize the construction below.

Definition IV.1. Let $M = (X, \Sigma, \delta, q_0)$ be as in Prop. III.3 with $\Sigma_{uc} \subseteq \Sigma$ and $\Gamma := \{\, \gamma \subseteq \Sigma \,|\, \Sigma_{\mathrm{uc}} \subseteq \gamma \,\}$. Then we define its associated game graph as $G(M) = (Q^0, Q^1, \delta^0, \delta^1, q_0)$ s.t.

- $Q^0 = X$
- $Q^1 = X \times \Gamma$
- $\delta^0(x) = \{x\} \times \Gamma$
- $x' \in \delta^1((x, \gamma))$ 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

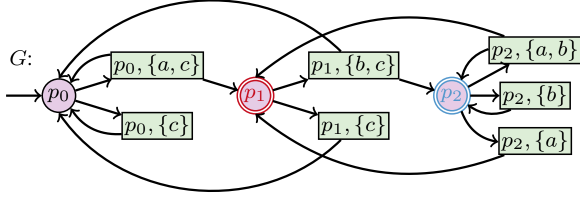

Fig. 5. Game graph $G(M)$ associated with the synthesis automaton M in Fig. 4. Supervisor and plant player states are indicated by a circular violet and a rectangular green shape, respectively. Rectangular states (p, γ) indicate the control pattern[5] γ chosen by the supervisor in state p of M.

0 states. I.e, the choice of control pattern γ in state $x \in X$ of M corresponds to the move of player 0 from $q = x$ to $q' = (x, \gamma)$ in G. Further, as M is assumed to have unique transition labels, this expansion allows to remove all transition labels resulting in an unlabeled game graph G as defined in Sec. IV-A. Fig. 5 shows the two-player game graph $G(M)$ corresponding to M in Fig. 4.

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 $(M, \mathcal{F}_P^{\mathsf{S}}, \mathcal{F}_S^{\mathsf{R}})$. 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

$$\forall \pi \in \mathcal{P}(M, \check{f}) \; . \; \left(\pi \in \mathcal{P}(M, \mathcal{F}_P^{\mathsf{S}}) \Rightarrow \pi \in \mathcal{P}(M, \mathcal{F}_S^{\mathsf{R}})\right), \quad (4)$$

which is in turn equivalent to

$$\forall \pi \in \mathcal{P}(M, \check{f}) \; . \; \left(\pi \notin \mathcal{P}(M, \mathcal{F}_P^{\mathsf{S}}) \vee \pi \in \mathcal{P}(M, \mathcal{F}_S^{\mathsf{R}})\right). \quad (5)$$

Consequently, (2a) is achieved by a supervisor $\check{f}$ which ensures plays over M either *do not* satisfy the Streett condition $\mathcal{F}_P^{\mathsf{S}}$ *or* fulfill the Rabin condition $\mathcal{F}_S^{\mathsf{R}}$. However, as Rabin and Streett conditions are duals, *not satisfying* the *Streett* condition $\mathcal{F}_P^{\mathsf{S}}$ is equivalent to *satisfying* the *Rabin* condition $\mathcal{F}_P^{\mathsf{R}} := \neg \mathcal{F}_P^{\mathsf{S}}$. Further, given the definition of Rabin winning conditions (see Sec. II), it is easy to see that a path over M satisfies either the Rabin condition $\mathcal{F}_P^{\mathsf{R}}$ *or* the Rabin condition $\mathcal{F}_S^{\mathsf{R}}$ iff it satisfies the Rabin condition $\mathcal{F}_{P\to S}^{\mathsf{R}} = \mathcal{F}_P^{\mathsf{R}} \cup \mathcal{F}_S^{\mathsf{R}}$. With this observation, we can further rewrite (2a) into the equivalent formula

$$\mathcal{P}(M, \check{f}) \subseteq \mathcal{P}(M, \mathcal{F}_{P\to S}^{\mathsf{R}}). \quad (6)$$

Thus, an obvious choice for the winning condition over the game graph $G(M)$ is the Rabin condition $\mathcal{F}_{P\to S}^{\mathsf{R}}$.

Example IV.1. Consider the example from Sec. III-E and recall that $\mathcal{F}_P^{\mathsf{S}} = \{\langle \{p_0, p_1, p_2\}, \{p_2\}\rangle\}$ and $\mathcal{F}_S^{\mathsf{R}} = \{\langle \{p_1\}, \emptyset \rangle\}$. This gives the Rabin winning condition

$$\mathcal{F}_{P\to S}^{\mathsf{R}} = \{\langle \{p_0, p_1, p_2\}, \{p_2\}\rangle, \langle \{p_1\}, \emptyset\rangle\} \quad (7)$$

for the induced game over $G(M)$. Intuitively, the condition in (7) states that either p_2 is only visited *finitely often* (first Rabin pair) or p_1 is visited *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 Sec. III-E). ◁

As the above example demonstrates, a winning strategy for $\mathcal{F}_{P\to S}^{\mathsf{R}}$ 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 non-conflicting 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 $(G, \mathcal{S}, \mathcal{W})$ where G is a game graph and $\mathcal{S}$ and $\mathcal{W}$ 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 $\mathcal{S}$ against any strategy of player 1 (the "system"), while *allowing* the system to cooperate with him to *additionally* fulfill $\mathcal{W}$. Such winning strategies are therefore called *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 $(G, \mathcal{S}, \mathcal{W})$, synthesize a strategy h for player 0 s.t.

(i) every play over G compliant with h is winning w.r.t. $\mathcal{S}$,

$$\mathcal{P}(G, h) \subseteq \mathcal{P}(G, \mathcal{S}) \quad (8a)$$

(ii) for every finite play ν over G compliant with h, there exists an infinite play ρ over G compliant with h and winning w.r.t. $\mathcal{W}$, s.t. $\nu \in \mathrm{pfx}(\rho)$, i.e.,

$$P(G, h) \subseteq \mathrm{pfx}\left(\mathcal{P}(G, h) \cap \mathcal{P}(G, \mathcal{W})\right), \quad (8b)$$

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 effectivly construct winning strategies of player 0 in such games, Thm. V.1 establishes that the same is true for obliging games.

Theorem V.1. *Every obliging game $(G, \mathcal{S}, \mathcal{W})$ is reducible to a two-player game with an ω-regular winning condition. In particular, an obliging game $(G, \mathcal{F}^{\mathsf{R}}, \mathcal{F}^{\mathsf{S}})$ with n states, a Rabin condition $\mathcal{F}^{\mathsf{R}}$ with k pairs, and a Streett condition $\mathcal{F}^{\mathsf{S}}$ with l pairs can be reduced to a two-player game with $nk^2k!2^{O(l)}$ states, a parity condition with $2k+2$ colors, and $2k2^{O(l)}$ memory.*

Proof. The first claim follows from [16, Thm.3]. For the second claim, recall that from a Streett condition with l pairs, one can construct a (non-deterministic) Büchi automaton with $2^{O(l)}$ 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 [27] with $2k$ colors. Now, the construction in [16, Lem.2, Thm.4] reduces an

[5] We restrict depicted control patterns to events enabled at the source state.

obliging game with $\tilde{n}$ 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 $O(\tilde{n}q)$ states, $2k+2$ colors, and memory $2qk$. Applying this reduction to our setting yields a parity game with $n \cdot k^2 \cdot k! \cdot 2^{O(l)}$ states, $2k+2$ colors, and memory $2k \cdot 2^{O(l)}$. ■

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 $\mathcal{S} := \mathcal{F}^{\mathsf{R}}_{P\to S}$ and $\mathcal{W} := \mathcal{F}^{\mathsf{S}}_{P}$ 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 $(M, \mathcal{F}^{\mathsf{S}}_{P}, \mathcal{F}^{\mathsf{R}}_{S})$ be a Streett/Rabin supervisor synthesis automaton and $G(M)$ its associated game graph. Then there exists a path-based supervisor $\check{f}$ that is a solution to the supervisor synthesis problem over $(M, \mathcal{F}^{\mathsf{S}}_{P}, \mathcal{F}^{\mathsf{R}}_{S})$ iff there exists a player 0 strategy h winning the obliging game $(G(M), \mathcal{F}^{\mathsf{R}}_{P\to S}, \mathcal{F}^{\mathsf{S}}_{P})$.*

In order to prove Thm. 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 vs. Plays. To formally connect paths in M to plays over G, we define the set-valued map $\mathsf{Plays} : x_0X^* \to 2^{q_0(Q^1Q^0)^*}$ iteratively as follows: $\mathsf{Plays}(x_0) := \{x_0\}$ and $\mathsf{Plays}(\nu x) := \{\mu\tilde{x}x \mid \mu \in \mathsf{Plays}(\nu), \tilde{x} \in \{\mathsf{Last}(\nu)\} \times \Gamma\}$. By slightly abusing notation, we extend the map Plays to infinite paths $\pi \in x_0X^\omega$ as the limit of all mappings $\mathsf{Plays}(p_n)$ where $(p_n) \in x_0X^*$ is the unbounded monotone sequence of prefixes of π. Similarly, we define the inverse map $\mathsf{Plays}^{-1} : q_0(Q^1Q^0)^* \to x_0X^*$ s.t. $\mathsf{Plays}^{-1}(\mu) = \nu$ where ν is the single element of the set $\{\nu \in x_0X^* \mid \mu \in \mathsf{Plays}(\nu)\}$. Again we extend Plays^{-1} to infinite strings in the obvious way.

The construction of $G(M)$ from M in Def. IV.1 allows us to show that the map 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 Prop. III.3 and $G(M)$ its associated game graph as in Def. IV.1. Then*

$$\mathsf{Plays}(P(M)) = P(G), \tag{9a}$$

$$\mathsf{Plays}(\mathcal{P}(M)) = \mathcal{P}(G), \text{ and} \tag{9b}$$

$$\mathsf{Plays}(\mathcal{P}(M,\mathcal{F})) = \mathcal{P}(G,\mathcal{F}), \tag{9c}$$

where $\mathcal{F}$ is a winning condition over M.

Proof. ► (9a): Let $\nu = x_0x_1 \ldots x_k \in P(M)$. Then $\mathsf{Plays}(\nu)$ is the set containing all plays $\mu := x_0(x_0,\gamma_0)x_1(x_1,\gamma_1)\ldots(x_k,\gamma_{k-1})x_k$ s.t. $\gamma_i \in \Gamma$ for all $i \in [0;k]$. It follows from Def. IV.1 that all $\mu \in \mathsf{Plays}(\nu)$ are a play over G starting in q_0, and, hence $\mathsf{Plays}(P(M)) \subseteq P(G)$. The inverse direction follows similarly from the last condition in Def. IV.1. ► (9b) follows directly from (9a) by taking the limit closure on both sides. ► (9b) First, any winning condition over M is also a winning condtion over G as $Q = Q^0 \cup Q^1$ with $Q^0 = X$.Now pick any path $\pi \in \mathcal{P}(M,\mathcal{F})$. Then we know that the set $\mathsf{Inf}(\pi) \subseteq X$ fulfills the conditions for acceptance w.r.t. the acceptance condition $\mathcal{F}$ over M. Now take any $\rho \in \mathsf{Plays}(\pi) \subseteq \mathcal{P}(G)$ and observe that deciding winning of ρ w.r.t. $\mathcal{F} \subseteq Q^0$ only depends on the set $\mathsf{Inf}(\rho)|_{Q^0} \subseteq Q^0$. Then the claim follows from the observation that the definition of Plays implies $\mathsf{Inf}(\rho)|_{Q^0} = \mathsf{Inf}(\pi)$. ■

Supervisors vs. 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 $G(M)$, every such play could in principle induce a different control decision. We call strategies that do not utilize this additional flexibility *non-ambiguous.*

Definition V.1. Let G be as in Def. IV.1. We call a player 0 strategy over G *non-ambiguous* if for any $\nu \in x_0X^*$ and any $\mu, \mu' \in \mathsf{Plays}(\pi)$, we have $\check{h}(\mu) = \check{h}(\mu')$.

A strategy over G can only choose one particular next state in a 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 non-ambiguous strategy $\check{h}$ over G from a given control strategy h with the same set of generated plays.

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

$$\check{h}(x_0) := h(x_0), \quad \check{h}(\mu\tilde{x}_kx_{k+1}) := h(\mu\check{h}(\mu)x_{k+1}) \tag{10}$$

is a non-ambiguous player 0 strategy over G and it holds that $\mathcal{P}(G,h) = \mathcal{P}(G,\check{h})$.

Proof. For the base-case, $\mathsf{Plays}(x_0) = \{x_0\}$ and therefore for all $\mu, \mu' \in \mathsf{Plays}(x_0)$ we have $\mu = \mu' = x_0$. Hence $\check{h}(\mu) = \check{h}(\mu') = h(x_0)$, i.e., $\mathcal{P}(G,h)|_{[0,0]} = \mathcal{P}(G,\check{h})|_{[0,0]}$. For the induction step, fix $\nu \in x_0X^*$ with $|\nu| = k > 1$ and assume that for all $\mu, \mu' \in \mathsf{Plays}(\nu)$ we have $\check{h}(\mu) = \check{h}(\mu')$. Now choose any $x \in X$ and observe that $\mathsf{Plays}(\nu x) = \{\mu\tilde{x}x \mid \mu \in \mathsf{Plays}(\nu), \tilde{x} \in \{\mathsf{Last}(\nu)\} \times \Gamma\}$. Now pick any two $\mu\tilde{x}x, \mu'\tilde{x}'x \in \mathsf{Plays}(\nu x)$ and observe that from the definition of $\check{h}$ follows that $\check{h}(\mu\tilde{x}x) = h(\mu\check{h}(\mu)x)$ and $\check{h}(\mu'\tilde{x}'x) = h(\mu'\check{h}(\mu')x)$. As $\mu, \mu' \in \mathsf{Plays}(\nu)$ it follows from the induction hypothesis that $\check{h}(\mu) = \check{h}(\mu')$ and therefore $\check{h}(\mu\tilde{x}x) = \check{h}(\mu'\tilde{x}'x)$. This proves that $\check{h}$ is non-ambiguous. Now assume $\Lambda := \mathcal{P}(G,h)|_{[0,k]} = \mathcal{P}(G,\check{h})|_{[0,k]}$ and $\check{h}(\mu) = h(\mu)$ for all $\mu \in \Lambda$. Then it follows from Def. IV.1 that

$\mathcal{P}(G,h')|_{[0,k+2]}$ contains all strings $\mu(\mathsf{Last}(\mu),\gamma)x'$ s.t. $\mu \in \Lambda$, $(\mathsf{Last}(\mu),\gamma) = h'(\mu)$ and $x' = \delta(x_0,\sigma)$ for some $\sigma \in \gamma$. With this it immediately follows from the induction hypothesis that $\mathcal{P}(G,h)|_{[0,k+2]} = \mathcal{P}(G,\check{h})|_{[0,k+2]}$. As both $\mathcal{P}(G,h)$ and $\mathcal{P}(G,\check{h})$ are closed languages, this proves the claim. ■

Prop. V.4 shows that restricting attention to non-ambiguous player 0 strategies over G is without loss of generality. Now it is easy to see that non-ambiguous 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 Thm. V.2 the following holds. (i) Let $\check{f}$ be a supervisor solving $(M, \mathcal{F}_P^{\mathsf{S}}, \mathcal{F}_S^{\mathsf{R}})$ and $\check{h}$ a player 0 winning strategy over $G(M)$ s.t.*

$$\forall \mu \in q_0(Q^1Q^0)^* \;.\; \check{h}(\mu) = (\mathsf{Last}(\mu), \check{f}(\mathsf{Plays}^{-1}(\mu))). \quad (11)$$

Then $\check{h}$ is a non-ambiguous winning strategy for $(G(M), \mathcal{F}^{\mathsf{R}}_{P\to S}, \mathcal{F}^{\mathsf{S}}_P)$. (ii) Let $\check{h}$ be a non-ambiguous winning strategy for $(G(M), \mathcal{F}^{\mathsf{R}}_{P\to S}, \mathcal{F}^{\mathsf{S}}_P)$ and $\check{f}$ s.t.

$$\check{f}(\nu) = \gamma \text{ with } \gamma \in \{\exists \mu \in \mathsf{Plays}(\nu) \;.\; \check{h}(\mu) = (\cdot, \gamma)\}. \quad (12)$$

Then $\check{f}$ is a path-based supervisor solving $(M, \mathcal{F}_P^{\mathsf{S}}, \mathcal{F}_S^{\mathsf{R}})$.

Proof. First, observe that given $\check{f}$, every $\check{h}$ fulfilling (11) is non-ambiguous by construction. Conversely, given a non-ambiguous strategy $\check{h}$, (12) implies that γ is uniquely defined for any $\nu \in x_0X^*$, i.e., $\check{f}$ is a path-based strategy over M. Further, given non-ambiguity of $\check{h}$ we can combine the induction from the proof of Prop. V.4 and the correspondence used in the proof of (9a) to conclude that

$$\mathcal{P}(G,\check{h}) = \mathsf{Plays}(\mathcal{P}(M,\check{f})). \quad (13)$$

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

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

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

C. Example

The technical reduction from obliging games to games with ω-regular winning conditions (see Thm. 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 $(G(M), \mathcal{F}^{\mathsf{R}}_{P\to S}, \mathcal{F}^{\mathsf{S}}_P)$ over the game graph $G(M)$ depicted in Fig. 5.

As the first step of this construction, we double the state space of G resulting in an upper and a lower part (see Fig. 6). The upper part is a copy of the old state space while in the lower part all states become control player states (indicated

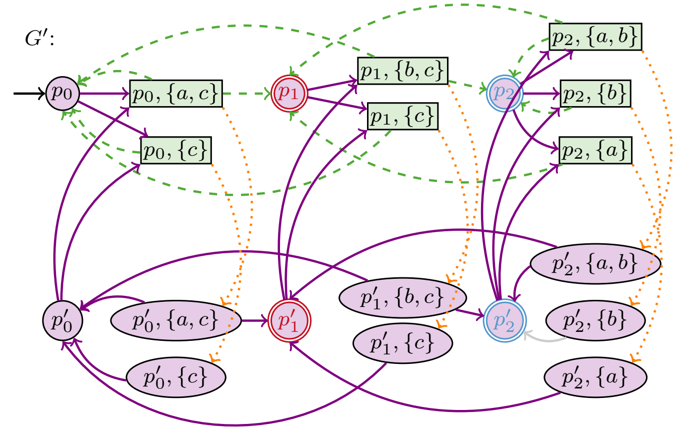

Fig. 6. Obliging game graph expansion of G in Fig. 5 as discussed in Sec. V-C (see [16] for a formalization). The plant can decide to chose the next event by herself (dashed green transitions) or to let the controller decide on her behalf (dotted orange transition followed by a solid violet one).

by their violet ellipse shape). Now we run the following Gedankenexperiment: in every (rectangular green) 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 γ and by this always moves to the rectangular green state (p_i,γ) 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 $\mathcal{F}^{\mathsf{R}}_{P\to S}$ in (7).

The bottom copy of the game makes sure that the resulting strategy is non-conflicting. 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., $\mathcal{F}^{\mathsf{S}}_P$) remains satisfiable along with satisfying $\mathcal{F}^{\mathsf{R}}_{P\to S}$. 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. It should be noted that the synthesis problem over the lower game graph is actually much simpler, as it only involves one player (namely the controller) and thereby reduces to a simple path search.

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. 6 separately. For the upper game, we know that a supervisor disabling events a and b in every state is winning w.r.t. $\mathcal{F}^{\mathsf{R}}_{P\to S}$ (see

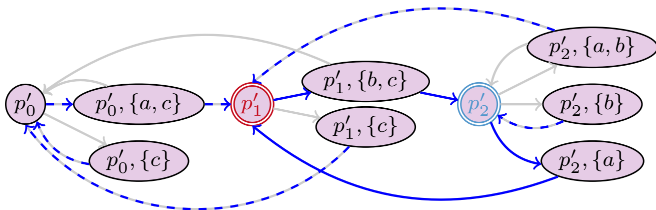

Fig. 7. Witness of a path from every reachable state of $G(M)$ (dashed) to a loop (solid) satisfying plant and specification makings always again. This defines a plant and control player strategy denoted by $g^{\downarrow}$ and $h^{\downarrow}$, respectively.

Example IV.1). Call this strategy $h^{\uparrow}$. We can assume w.l.o.g. that $h^{\uparrow}$ is memoryless[6], 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. 7, indicating an infinite trace from every state visiting both p_1 and p_2 infinitely often, fulfilling both $\mathcal{S} = \mathcal{F}^{\mathtt{R}}_{P\rightarrow S}$ and $\mathcal{W} = \mathcal{F}^{\mathtt{S}}_{P}$. This path immediately defines a memoryless plant and a control player strategy which we denote $g^{\downarrow}$ and $h^{\downarrow}$, respectively.

Given $h^{\uparrow}$, $g^{\downarrow}$, and $h^{\downarrow}$, we can combine them into a solution to the original synthesis problem over $G(M)$ (and therefore M) in Fig. 5, 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 $g^{\downarrow}$ ($m = 1$) or not ($m = 0$). If $m = 1$, the controller executes the unique pattern chosen by $h^{\downarrow}$ in the next state. Otherwise, it operates according to $h^{\uparrow}$.

For the particular choices of strategies in this example we see that the only allowed event c in (p_0, γ) is part of $g^{\downarrow}$ and therefore triggers $h^{\downarrow}$. Hence, the actual closed loop allows the plant to move to p_1 next. If it does so, $h^{\downarrow}$ remains active as this move is again contained in $g^{\downarrow}$ (see Fig. 7). If the plant decides to stay in p_0, $h^{\uparrow}$ 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 Thm. III.4 and Thm. V.2 enables us to solve a given supervisory synthesis problem (Problem 1) over a plant model $(L_P, \mathcal{L}_P)$ w.r.t. a specification $\mathcal{L}_S$ and a set of uncontrollable events $\Sigma_{uc} \subseteq \Sigma$ through the following steps:

1) Construct a Street/Rabin synthesis automaton $(M, \mathcal{F}^{\mathtt{S}}_{P}, \mathcal{F}^{\mathtt{R}}_{S})$ as in Prop. III.3.
2) Extend M into a game graph $G(M)$ as in Def. IV.1.
3) Solve the obliging game $(G(M), \mathcal{F}^{\mathtt{R}}_{P\rightarrow S}, \mathcal{F}^{\mathtt{S}}_{P})$ via its reduction to standard ω-regular games (see Thm. V.1).
4) If the obliging game has no solution, also Problem 1 has no solution (see Thm. III.4 and Thm. V.2).
5) If the obliging game allows for a control strategy h, compute its induced non-ambiguous strategy $\check{h}$ as in (10).
6) Reduce $\check{h}$ to a path-based supervisor via (11), which in turn defines the event-based supervisor f via (3).
7) Then f solves Problem 1 (see Thm. III.4 and Thm. V.2).

The complexity of this algorithm can be derived from Thm. V.1 in the following way. Given a finite state machine M with n states we get a game graph $G(M)$ with $n2^{|\Sigma_c|}$ states. Further, given the Streett and Rabin conditions $\mathcal{F}^{\mathtt{S}}_{P}$ and $\mathcal{F}^{\mathtt{R}}_{S}$ with l and k pairs, we get an obliging game having a strong Rabin condition with $l + k$ pairs and a weak Streett condition with l pairs. Finally, a parity game with $\tilde{n}$ states and $\tilde{k}$ colors can be solved in $O(\tilde{n}^{\tilde{k}})$ time. Hence, our solution can be computed in time $O((n2^{|\Sigma_c|}(l+k)^2(l+k)!2^{O(l)})^{2(l+k)+2})$. 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., $\mathcal{L}_P = \Sigma^{\omega}$) as solving Rabin games is NP-complete [29]. 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 [24] for the special case where $\mathcal{F}_P$ and $\mathcal{F}_S$ are each a generalized Büchi winning condition. We postpone the generalization of this algorithm to future work. .

[6]A strategy $h : q_{init}(Q^1Q^0)^* \rightarrow Q^1$ is memory-less if for all $\nu, \nu' \in (Q^0Q^1)^*$ and $q \in Q^0$ holds that $h(\nu q) = h(\nu' q)$. I.e., the strategy bases its choice of control patterns purely on the current state of the play.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control and Optimization*, vol. 25, pp. 206–230, 1987.

[2] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

[3] W. M. Wonham and K. Cai, *Supervisory control of discrete-event systems*, ser. Communications and Control Engineering. Springer, 2019.

[4] J. G. Thistle and W. M. Wonham, "Control of ω-automata, church's problem, and the emptiness problem for tree ω-automata," in *Int. Workshop on Computer Science Logic*. Springer, 1991, pp. 367–381.

[5] ——, "Supervision of infinite behavior of discrete-event systems," *SIAM J. on Control and Optimization*, vol. 32, no. 4, pp. 1098–1113, 1994.

[6] ——, "Control of infinite behavior of finite automata," *SIAM J. on Control and Optimization*, vol. 32, no. 4, pp. 1075–1097, 1994.

[7] J. G. Thistle, "On control of systems modelled as deterministic rabin automata," *Discrete Event Dyn. Systems*, vol. 5, no. 4, pp. 357–381, 1995.

[8] A. Church, "Applications of recursive arithmetic to the problem of circuit synthesis," *Summaries of the Summer Institute of Symbolic Logic, Volume I*, pp. 3–50, 1957.

[9] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *Annual symposium on theoretical aspects of computer science*. Springer, 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," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 985–1010, 2000.

[12] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi, "Supervisory control and reactive synthesis: a comparative introduction," *Discrete Event Dynamic Systems*, 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 *21st IFAC World Congress*, 2020.

[14] Z. Ramezani, J. Krook, Z. Fei, M. Fabian, and K. Akesson, "Comparative case studies of reactive synthesis and supervisory control," in *2019 18th European Control Conference (ECC)*. IEEE, 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 Transactions on Automatic Control*, vol. 65, no. 8, pp. 3502–3516, 2019.

[16] K. Chatterjee, F. Horn, and C. Löding, "Obliging games," in *Int. Conference on Concurrency Theory*. Springer, 2010, pp. 284–296.

[17] A.-K. Schmuck, T. Moor, and R. Majumdar, "On the relation between reactive synthesis and supervisory control of non-terminating processes," *Discrete Event Dynamic Systems*, vol. 30, pp. 81–124, 2020.

[18] G. Aucher, "Supervisory control theory in epistemic temporal logic," in *AAMAS '14*, 2014, pp. 333–340.

[19] A. C. van Hulst, M. A. Reniers, and W. J. Fokkink, "Maximally permissive controlled system synthesis for non-determinism and modal logic," *Discrete Event Dynamic Systems*, vol. 27, no. 1, pp. 109–142, Mar 2017.

[20] A. Sakakibara and T. Ushio, "Hierarchical control of concurrent discrete event systems with linear temporal logic specifications," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101.A, no. 2, pp. 313–321, 2018.

[21] J.-M. Yang, T. Moor, and J. Raisch, "Refinements of behavioural abstractions for the supervisory control of hybrid systems," *Discrete Event Dynamic Systems*, pp. 1–28, 2020.

[22] C. Baier and T. Moor, "A hierarchical control architecture for sequential behaviours," *IFAC Proceedings Volumes*, vol. 45, no. 29, pp. 259–264, 2012.

[23] T. Moor, "Supervisory control on non-terminating processes: An interpretation of liveness properties," Lehrstuhl für Regelungstechnik, Friedrich-Alexander Universität Erlangen-Nürnberg, Tech. Rep., 2017.

[24] R. Majumdar, N. Piterman, and A.-K. Schmuck, "Environmentally-friendly GR(1) synthesis," in *TACAS'20*. Springer, 2019, pp. 229–246.

[25] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distributed computing*, vol. 2, no. 3, pp. 117–126, 1987.

[26] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 3rd ed. Springer, 2021.

[27] E. Emerson and C. Jutla, "Tree automata, mu-calculus and determinacy," in *FOCS'91*, Oct 1991, pp. 368–377.

[28] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1989, pp. 179–190.

[29] E. A. Emerson and C. S. Jutla, "The complexity of tree automata and logics of programs," in *FoCS*, vol. 88. Citeseer, 1988, pp. 328–337.

[30] W. Thomas, "On the synthesis of strategies in infinite games," in *STACS'95 Munich, Germany*, 1995, pp. 1–13.