A Reactive Synthesis Approach to Supervisory Control of Terminating Processes

Anne-Kathrin Schmuck [*] **Thomas Moor** [**] **Klaus Werner Schmidt** [***]

[*] *Max Planck Institute for Software Systems, Kaiserslautern, Germany (e-mail: akschmuck@mpi-sws.org)*
[**] *Lehrstuhl für Regelungstechnik, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany (e-mail: lrt@fau.de)*
[***] *Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey (e-mail: schmidt@metu.edu.tr)*

Abstract: This paper establishes a connection between supervisory control theory (SCT) and reactive synthesis (RS) in the situation where both the plant and the specification are modeled by $*$-languages, i.e., formal languages over *finite* words. In particular, we show that the deterministic finite automaton G typically used in SCT to construct a *maximally permissive supervisor* f for a plant language L w.r.t. a specification language E, can be interpreted as a two-player game which allows to solve the considered synthesis problem by a two-nested fixed-point algorithm in the μ-calculus over G. The resulting game turns out to be a cooperative Bchi-type game which allows for a maximally permissive solution in the particular context of SCT. This is surprising, as classical Bchi games do not have this property.

Keywords: Discrete-event systems, supervisory control, reactive synthesis, μ-calculus, game theory.

INTRODUCTION

Supervisory control theory (SCT) is a branch of control theory that addresses the systematic design of controllers for *discrete-event systems* — dynamical systems whose behavior can be modeled by discrete sequences of so called *events*, which model a particular value change in a relevant variable of the considered system. Typically, the set of such events is considered finite and all event sequences a system (typically called the *plant*) can possibly exhibit, are collected in a formal language L functioning as the plant model. On the other hand, a *specification language* E contains only *desired* event sequences the plant *should* exhibit. The controller synthesis problem then asks to construct a so-called *supervisor* f which appropriately restricts the evolution of the plant s.t. all resulting event sequences respect the specification. Supervisory control theory was originally proposed by Ramadge and Wonham (1987) and now is an established field of research, with standard text books by Cassandras and Lafortune (2008) and Wonham and Cai (2019), as well as recent reviews by Wonham et al. (2018) and Lafortune (2019) summarizing the current state-of-the-art in this field.

Within this paper we continue recent efforts, e.g. by Ehlers et al. (2017), Ramezani et al. (2019) and Schmuck et al. (2020), to draw formal connections between SCT and *reactive synthesis* (RS), a branch of computer science. RS aimes at *synthesizing* a computer program which operates in accordance to a prescribed specification while *reacting* to external environment inputs. For a comprehensive introduction to RS and its connection to control see e.g. Finkbeiner (2016) and Maler (2002), respectively.

Our continuation of these comparative efforts within this paper is motivated by recent successes stories of the use of comparative insights for solving long-standing open problems both in SCT and RS. On one hand, it has been shown by Yin and Lafortune (2016a) that most permissive supervisors for plants under partial observation can be constructed by utilizing a game-based approach inspired by RS. This insight has also been used to enforce opacity (Hélouët et al., 2018) or fault diagnosability (Cassez and Tripakis, 2008; Yin and Lafortune, 2016b), or to optimize sensor activation for these synthesis problems (Yin and Lafortune, 2019). Conversely, it had been shown by Schmuck et al. (2020); Majumdar et al. (2019) that transferring ideas from SCT to RS results in algorithms which handle environment assumptions more satisfactory in the context of cyber-physical-system design.

It is the aim of this paper to further facilitate the connection between SCT and RS to enhance interesting developments within their intersection. In particular, we show that the connection between SCT and RS established for *non-terminating* plant and specification processes modelled by regular ω-languages in Schmuck et al. (2020) carries over to terminating processes, i.e., plants and specifications modelled by regular $*$-languages.

In contrast to Schmuck et al. (2020), however, we show this connection on the algorithmic level. Given the most basic supervisor synthesis problem defined by $*$-languages L and E which are generated by deterministic finite automata G_L and G_E, we follow the common approach (see e.g., Cassandras and Lafortune (2008)) to construct a particular product automaton G to synthesize the supervisor f. We then interpret the supervisor synthesis problem over G as a two player game, in which the supervisor (player 0) applies a control pattern (i.e., provides a set of enabled events) and in which the plant (player 1) subsequently chooses to append an enabled event to resolve the remaining non-determinism. With this interpretation, we show that there exists a fixed-point in the μ-calculus over G

which computes the so-called *winning states* of G, that is, all states of G which allow for a supervisor which restricts the plant appropriately, such that all generated event sequences fulfill the specification. The obtained set of winning states then allows us to extract the desired maximally permissive supervisor f for the control problem described by L and E if and only if the initial state of G is contained in the winning state set.

Our work is closely related to supervisor synthesis methods using a game-theoretic interpretation, e.g. by Yin and Lafortune (2016a); Hélouët et al. (2018); Cassez and Tripakis (2008), as discussed above. While these works utilize a game-based approach to compute supervisors for special instances of SCT, our contribution has a more tutorial flavor. For the most basic setting of SCT, we derive the winning conditions and symbolic synthesis procedures à la RS, which capture the exact intend of SCT. In particular, we show that supervisor synthesis corresponds to solving a cooperative Büchi-type game over G. Interestingly, this modified Büchi game allows for a maximally permissive control strategy, which is not true for standard Büchi games. This comparative result therefore shows, that an SCT interpretation of control helps to relax synthesis problems s.t. maximally permissive solutions are available.

Our established connection between the synthesis algorithms for solving supervisor and reactive synthesis problems via Büchi-type games extends the connection derived in Ehlers et al. (2017). There, the authors show that the supervisor synthesis problem over G (as described above) corresponds to a particular specification in computational tree logic (CTL). Based on this result, they show that one can transfer G into a so called Kripke-structure which allows to apply standard techniques from RS to solve the synthesis problem w.r.t. the derived CTL specification. Then, they validate that the obtained result indeed solves the posed SCT problem. Our approach, however, does not transfer the synthesis problem into a different modeling formalism to apply RS techniques. We rather show that the supervisor synthesis problem over G coincides with a cooperative Büchi-type game à la RS over G.

This paper is by far not the first to combine supervisory controller synthesis with fixed-points in the μ-calculus. There has been notable work on synthesizing supervisors w.r.t. temporal specifications, e.g. given in linear temporal logic (LTL) by Sakakibara and Ushio (2018), computational tree logic (CTL) by Jiang and Kumar (2006), epistemic temporal logic by Aucher (2014) or modal logic by van Hulst et al. (2017). All these works adapt the powerful machinery of reactive synthesis to ensure that computed strategies respect the two fundamental requirements of SCT, namely controllability and non-blockingness, on top of enforcing the imposed temporal specification. In contrast to this, our paper derives the fixed point which exactly captures the "common" intent of SCT and works directly over G.

1. PRELIMINARIES

For a general introduction to supervisory control see (Cassandras and Lafortune, 2008). We give a summary of common notation and elementary facts relevant for the present paper.

Notation. Let Σ be a *finite alphabet*, i.e., a finite set of symbols $\sigma \in \Sigma$. The *Kleene-closure* Σ^* is the set of finite strings $s = \sigma_1\sigma_2\cdots\sigma_n$, $n \in \mathbb{N}$, $\sigma_i \in \Sigma$, and the *empty string* $\epsilon \in \Sigma^*$, $\epsilon \notin \Sigma$. The length of a string $s \in \Sigma^*$ is denoted $|s| \in \mathbb{N}_0$, with $|\epsilon| = 0$. If, for two strings $s, r \in \Sigma^*$, there exists $t \in \Sigma^*$ such that $s = rt$, we say r is a *prefix* of s, and write $r \leq s$.

Formal Languages. A *language* over Σ is a subset $L \subseteq \Sigma^*$. The *prefix* of a language $L \subseteq \Sigma^*$ is defined by $\mathrm{pfx}(L) := \{r \in \Sigma^* \mid \exists s \in L : r \leq s\}$. The prefix operator is also referred to as the *prefix-closure*, and, a language L is *closed* if $L = \mathrm{pfx}(L)$. A language K is *relatively closed w.r.t. L* if $K = \mathrm{pfx}(K) \cap L$. The prefix operator distributes over arbitrary unions of languages. However, for the intersection of two languages L and M, we have $\mathrm{pfx}((L \cap M)) \subseteq (\mathrm{pfx}(M)) \cap (\mathrm{pfx}(M))$. If equality holds, L and M are said to be *non-conflicting*.

Automata. An *automaton* is a tuple $G = (Q, \Sigma, \delta, q_{\mathrm{init}}, Q_{\mathrm{m}})$, with *state set* Q, *initial state* $q_{\mathrm{init}} \in Q$, *marked states* $Q_{\mathrm{m}} \subseteq Q$, and the partial *transition function* $\delta : Q \times \Sigma \rightarrow Q$. We write $\delta(q, s)!$ to indicate that δ is defined for the specified arguments $q \in Q$ and $s \in \Sigma^*$. The automaton is *full* if $\delta(q, s)!$ for all $q \in Q$ and $s \in \Sigma^*$, i.e., if δ is a proper function. With the automaton $G = (Q, \Sigma, \delta, q_{\mathrm{init}}, Q_{\mathrm{m}})$, we denote the *generated language* $\mathbf{L}(G) := \{ s \in \Sigma^* \mid \delta(q_{\mathrm{init}}, s)! \}$ and the *marked language* $\mathbf{L}_m(G, Q_{\mathrm{m}}) := \{ s \in \Sigma^* \mid \delta(q_{\mathrm{init}}, s) \in Q_{\mathrm{m}}) \}$. We omit the parameter Q_{m} in the definition of $\mathbf{L}_m()$ if it is clear from the context.

Fixpoint Calculus. Let Q denote a finite set and consider a *monotone operator* f, i.e., $f(P') \subseteq f(P'') \subseteq Q$ for all $P' \subseteq P'' \subseteq Q$. Then the least fixpoint of f is given by $\cup\{f^i(\emptyset) \mid i \in \mathbb{N}\}$ and can be obtained by the iteration $P_1 := \emptyset$, $P_{i+1} := P_i \cup f(P_i)$. In particular, the fixpoint is attained for some finite $i \in \mathbb{N}$. Likewise, the iteration $P_1 := Q$, $P_{i+1} := P_i \cap f(P_i)$ can be used to obtain the greatest fixpoint. As a μ-calculus formula, the least fixpoint of f is denoted $\mu P.f(P)$, whereas the greatest fixpoint is denoted $\nu P.f(P)$. Now consider an operator g that depends on multiple set-valued parameters, e.g., $g(P', P'') \subseteq Q$ for $P', P'' \subseteq Q$. Assuming that g is monotone in its first argument, the formulas $\mu P'.g(P', P'')$ and $\nu P'.g(P', P'')$ are well defined. Provided that g is also monotone in its second argument, the respective fixpoints are monotone in P''. In this case, nested μ-calculus formulae are well defined, e.g. $\nu P''.\mu P'.g(P', P'')$ evaluates to the greatest fixpoint of $\mu P'.g(P', P'')$, interpreted as an expression in terms of P''.

2. THE SUPERVISOR SYNTHESIS PROBLEM

Before defining the common supervisor synthesis problem formally in Sec. 2.2, we give an intuitive introduction to supervisory control via an illustrative example.

2.1 Illustrative Example

Consider a regular plant $L \subseteq \Sigma^*$, realized by the automaton $G_L = (P, \Sigma, \delta_L, p_0, P_m)$ with $\mathbf{L}_m(G_L) = L$, depicted in Fig. 1 (top left). It models a process which can take a controllable (i.e., preventable) action c (indicated by a tick on the respective arrow in Fig. 1) or an uncontrollable (i.e., un-preventable) action u (indicated by normal transition arrows in Fig. 1) in every state, giving the alphabet $\Sigma = \{u, c\}$. States p_3, p_4 and p_8 are marked in G_L (i.e., $P_m = \{p_3, p_4, p_8\}$) indicated by a double boundary line in Fig. 1. Given that L is a $*$-language containing possibly arbitrary long but *finite* event sequences, marked states indicate that the process modeled by G_L can (and eventually will) safely terminate its operation when in any of these marked states. In turn, this means that it can (and eventually will) *not*

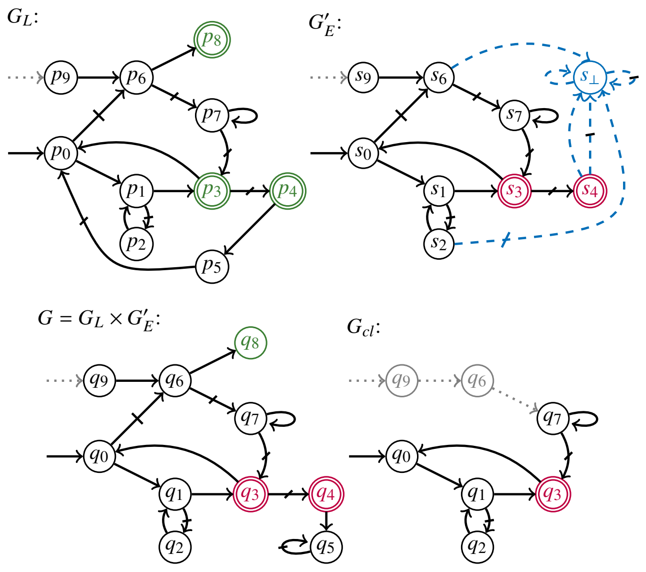

Fig. 1. Top: Automata representations G_L of the plant language L (top left) and G_E of the specification language E (top right, black) with its full extension G'_E (top right, black and blue). Bottom: Product automaton $G = G_L \times G'_E$ used for synthesizing the supervisor f (bottom left) and its restriction G_{cl} to its *winning states* used to extract f. Marked states have a double boundary and event labels from $\Sigma = \{c, u\}$ are omitted. Uncontrollable transitions on u are indicated by a tick on the arc; controllable transitions c are indicated by a normal arc.

reside in any other state infinitely long. Given this interpretation of a plant model, the automaton G_L should never be blocking. If it is, it means that the process can terminate its operation in an un-marked state, which should be made explicit by marking this terminal state. However, our developments in this paper do formally not require G_L to be non-blocking.

Similarly, consider the regular specification E, realized by the automaton $G_E = (S, \Sigma, \delta_L, s_0, S_m)$ with $\mathbf{L}_m(G_E) = E$, depicted in Fig. 1 (top right, black/purple). Intuitively, this specification asks us to ensure that the plant at hand only terminates its operation in s_3 and s_4, and to only continue operating (by returning to the initial state s_0), when in s_3. Whenever the plant reaches s_4, it should immediately terminate its operation.

To synthesize a supervisor enforcing this specification on the given plant, one typically constructs a particular product automaton by first completing G_E to a full automataton G'_E with generated language $\mathbf{L}(G'_E) = \Sigma^*$ and accepted language $\mathbf{L}_m(G'_E) = E$. This is done by adding a sink state (called $s_\perp$ in Fig. 1 (top right, blue)) and re-directing all non-enabled transitions in G_E to this state (see Fig. 1 (top right, blue, dashed)). After this, the usual product automaton G of G_L and G'_E is constructed, depicted in Fig. 1 (bottom left). For simplicity, we have renamed state pairs (p_i, s_i) and $(p_i, s_\perp)$ in the constructed product automaton to q_i.

Given the underlying plant dynamics modeled by G_L, we know that the plant will eventually transition to its markings when in an unmarked state. Given this intuition, the controller only has to ensure that the plant always remains in the co-reachable part of G while respecting uncontrollable transitions. That is, the supervisor can never prevent the plant from taking an un-controllable (i.e., un-preventable) transition in G. By inspection, we see that states $q_0 - q_4$ and q_7 allow a restriction of enabled events s.t. the plant eventually transitions to (or already is in) q_3 and q_4. It is however not true for q_6, as the supervisor cannot disable uncontrollable events and therefore cannot prevent the plant to transition to q_8 from q_6, which renders any marked state in G unreachable. Further, it should be noted that, once in q_4 the plant can transition to q_5 which also renders all marked states of G unreachable. As this transition is uncontrollable, q_4 does not allow to properly control the plant, which is only true for states $q_0 - q_3$ and q_7, which we term "winning states" of G. When restricting G to its winning state set, we obtain the automaton G_{cl} depicted in Fig. 1 (bottom right, black/purple).

When applying standard supervisory controller synthesis algorithms (see e.g. (Cassandras and Lafortune, 2008, p.168)) to this example, we would additionally remove q_7 from G_{cl}, as this state is not reachable in the constructed sub-automaton. Intuitively, G_{cl} realizes the closed-loop behavior of the plant, i.e., $\mathbf{L}_m(G_{cl})$ contains all event sequences the plant produces when supervised in the sense discussed before.

Now let us assume that p_2 is also marked in G_L, while s_2 stays unmarked in G_E. Then, by the usual automata product, q_2 will also stay unmarked both in G and in G_{cl}. This, however, implies that the plant under supervision cannot (and will never) reside in q_2 forever and therefore no finite string ending in q_2 is part of $\mathbf{L}_m(G_{cl})$. This, however, is counter intuitive given that G_L models the uncontrolled plant behavior correctly, i.e., stating that the plant can (and eventually might) terminate its operation in p_2 (and therefore q_2). As the supervisor has no means to enforce enabled transitions, the constructed closed loop behavior will turn out incorrect.

This problem is typically circumvented by requiring that the specification language E is closed w.r.t. L, that is, $E = \text{pfx}(E) \cap L$. Loosely speaking, this requires that G_E's marking respects G_L's marking, i.e., for any pair (p_i, s_i) in the product automaton either both states are marked or none (which is not fulfilled if p_2 is marked). In particular, relative closedness of E implies $E \subseteq L$ and, thus, $L_m(G) = E$. I.e., the specification can remove a marking from G_L by removing all traces ending in this state from the marked language (which is the case for $(p_8, s_\perp)$). Consistent with the original literature, we therefore restrict attention to relatively closed specification languages.

To conclude the example, let us assume that p_9, s_9 and therefore q_9 are the initial states of the respective automata in Fig. 1 (indicated by the dotted gray arrow pointing at them). Then one can verify that the synthesis problem at hand has no solution as q_6 does not allow to control the system as desired. Therefore, existing synthesis algorithms will immediately terminate with an empty closed-loop system G_{cl} after investigating q_6. However, it is easy to see that a change in the initial state does not change the general reasoning about states $q_0 - q_3$ and q_7 allowing to suitably control the plant. If, for example, the transition from q_6 to q_8 is a rare event that only happens if the machine at hand needs to be maintained, we can assume that for all interesting operational modes the plant will transfer from q_6 to q_7. Then knowing that q_7 allows to control the plant as desired allows us to provide a controller in this case.

The remainder of this paper formalizes the computation of the set of states of G that we have termed "winning state set". We show that there exists a two-nested fixed-point algorithm inspired by reactive synthesis which achieves this goal.

2.2 Problem Statement

The *supervisory synthesis problem*, first stated by Ramadge and Wonham (1987), is defined by a *plant* language $L \subseteq \Sigma^*$ modeling the process to be controlled, and a *specification* language $E \subseteq \Sigma^*$ s.t. E is relatively closed w.r.t. L, i.e., $E = \mathrm{pfx}(E) \cap L$. The common alphabet Σ of L and E is further partitioned in controllable and uncontrollable events, $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. Given this partition, one defines the set of *control patterns* $\Gamma \subseteq \Sigma$, s.t.

$$\Gamma := \{ \gamma \subseteq \Sigma \,|\, \Sigma_{uc} \subseteq \gamma \}. \tag{1}$$

Given a plant $L \subseteq \Sigma^*$, a *supervisor* is defined as a map $f : \mathrm{pfx}(L) \to \Gamma$ that maps event sequences $s \in \mathrm{pfx}(L)$ to a control pattern $f(s) \in \Gamma$, where the latter specifies the set of enabled successor events after the occurrence of s. A word $s \in \mathrm{pfx}(L)$ is called *consistent* with f if for all $\sigma \in \Sigma$ and $t\sigma \in \mathrm{pfx}(s)$ holds that $\sigma \in f(t)$. We collect all words of $\mathrm{pfx}(L)$ consistent with f in the set $\mathbf{L}(\mathrm{pfx}(L), f)$ which is non-empty and closed.

With this, the supervisory controller synthesis problem can be stated as follows.

Problem 1. Given a plant L and a relatively closed specification $\emptyset \neq E \subseteq L$, synthesize a supervisor $f : \mathrm{pfx}(L) \to \Gamma$ s.t.

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

$$\mathbf{L}(\mathrm{pfx}(L), f) \cap L \subseteq E \tag{2a}$$

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

$$\mathbf{L}(\mathrm{pfx}(L), f) = \mathrm{pfx}(\mathbf{L}(\mathrm{pfx}(L), f) \cap L), \tag{2b}$$

or determine, that no such supervisor exists.

As $\supseteq$ in (2b) always holds, the constraint is imposed by $\subseteq$ and ensures that the plant is always able to generate events allowed by f s.t. it generates a word in its marked language L. Then, by requirement (2a), all such generated words must be contained in the upper bound specification E.

Except for the avoidance of an explicit reference to an automaton representation of L, Problem 1 is equivalent to the synthesis problem studied by Ramadge and Wonham (1987). From the given reference we recall that if a solution exists there also exists a *maximally permissive* one; i.e. a supervisor $f : \mathrm{pfx}(L) \to \Gamma$ that solves Problem 1, such that $\mathbf{L}(\mathrm{pfx}(L), f') \subseteq \mathbf{L}(\mathrm{pfx}(L), f)$ for all other solutions $f' : \mathrm{pfx}(L) \to \Gamma$. Although maximal permissiveness is not the main concern in our discussion, we will establish this property for supervisors obtained by our alternative synthesis procedure.

2.3 Automata Realizations for Synthesis

In the basic setting of supervisory control theory discussed by Ramadge and Wonham (1987), both L and E are assumed to be realizable by deterministic finite automata $G_L = (P, \Sigma, \delta_L, p_0, P_m)$ and $G_E = (S, \Sigma, \delta_E, s_0, S_m)$, respectively. Then one can apply the method discussed in Sec. 2.1 to first extend G_E to the full automaton G'_E and then construct the product automaton $G = G_L \times G'_E = (Q, \Sigma, \delta_L, q_{\mathrm{init}}, Q_{\mathrm{m}})$. As E is assumed to be non-empty and relatively closed w.r.t. L, one obtains $\emptyset \neq E \subseteq L$ and, hence, $\mathbf{L}_m(G) = \mathbf{L}_m(G_L \times G'_E) = L \cap E = E$ and $\mathbf{L}(G) = \mathbf{L}(G_L \times G'_E) = \mathrm{pfx}(L) \cap \Sigma^* = \mathrm{pfx}(L)$.

The automaton G is typically used to compute the maximally permissive supervisor f that solves Problem 1 via the algorithm, e.g., presented in Cassandras and Lafortune (2008). It is known that such an f exists if and only if the standard construction returns a non-empty sub-automaton of G.

For our subsequent development of an algorithm to compute the set of *winning states* to derive f instead, it will turn out convenient to memorize two sets of marked states in G, namely $F_L := P_m \times S \subseteq Q$ and $F_E := P \times S_m \subseteq Q$. We denote by $\mathbf{L}_m(G, F_L)$ and $\mathbf{L}_m(G, F_E)$ the language marked by G when interpreting F_L and F_E as the set of marked states, respectively. It is easy to verify that $\mathbf{L}_m(G, F_L) = L$ and $\mathbf{L}_m(G, F_E) = E$. As E is assumed to be relatively closed w.r.t. L we further have $F_E \subseteq F_L$ and $F_E = Q_m$.

3. A GAME-THEORETIC PERSPECTIVE

3.1 Supervisory Control as a Two-Player Game

In view of reactive synthesis, we interpret the interaction of a supervisor and the plant as a two-player game played over G. Here, 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. In this context, we define a player 0 strategy (or *control strategy*) as a function $h : q_{init}Q^* \to \Gamma$; it is *memoryless* if $h(\nu q) = h(q)$ for all $\nu \in Q^*$ and all $q \in Q$. A *strategy* for player 1 (or *plant strategy*) is defined as a function $g : q_{init}Q^*\Gamma \to \Sigma$ s.t. $g(\nu\gamma) \in \gamma$ for all $\nu \in q_{init}Q^*$. The sequence $\pi = q_0\sigma_0 q_1\sigma_1 \ldots q_k$ is called a play over G if $q_0 = q_{init}$ and for all $i \in [1; k]$ it holds that $q_i = \delta(q_{i-1}, \sigma_{i-1})$; π is *consistent* with a control strategy h if it additionally holds that $\sigma_i \in h(q_0 \ldots q_{i-1})$. We denote by $\mathbf{P}(G)$ and $\mathbf{P}(G, h)$ the set of all plays over G and the set of plays consistent with h, respectively. Further, we denote by $\mathbf{L}(G, h)$ the set of all strings $s = \sigma_0 \ldots \sigma_k$ s.t. there exists $\pi = q_0\sigma_0 q_1\sigma_1 \ldots q_{k+1} \in \mathbf{P}(G, h)$.

As G is deterministic we have a one-on-one correspondence between state sequences and event sequences, that is, we have a one-on-one function $\mathbf{s} : q_{init}Q^* \to \mathrm{pfx}(L)$, whose inverse $\mathbf{s}^{-1} : \mathrm{pfx}(L) \to q_{init}Q^*$ is also one-on-one. This induces a one-on-one correspondence between the supervisor $f : \mathrm{pfx}(L) \to \Gamma$ in Problem 1 and the control strategy $h : q_{init}Q^* \to \Gamma$ which is summarized in the following lemma.

Lemma 2. Let $h : q_{init}Q^* \to \Gamma$ and define $f := h \circ \mathbf{s}^{-1}$, then $\mathbf{L}(G, h) = \mathbf{L}(\mathrm{pfx}(L), f)$. Conversely, let $f : \mathrm{pfx}(L) \to \Gamma$ and define $h : f \circ \mathbf{s}$. Then $\mathbf{L}(G, h) = \mathbf{L}(\mathrm{pfx}(L), f)$.

Proof. We pick h and $f = h \circ \mathbf{s}^{-1}$ and show $\mathbf{L}(G, h) \subseteq \mathbf{L}(\mathrm{pfx}(L), f)$: Pick $s = \sigma_0 \ldots \sigma_{k-1} \in \mathbf{L}(G, h)$ and recall that this implies that there exists $\pi = q_0\sigma_0 \ldots \sigma_{k-1}q_k \in \mathbf{P}(G, h)$ s.t. for all $i \in [0, k-1]$ holds $\sigma_i \in h(q_0, \ldots, q_i)$. As G is deterministic we further have that $q_0 \ldots q_i = \mathbf{s}^{-1}(\sigma_0 \ldots \sigma_{i-1})$. This implies $\sigma_i \in h(\mathbf{s}^{-1}(\sigma_0 \ldots \sigma_{i-1})) = f(\sigma_0 \ldots \sigma_{i-1})$ for all $i \in [0, k-1]$ where obviously $\sigma_0 \ldots \sigma_{i-1} \in \mathrm{pfx}(s)$. This implies $s \in \mathbf{L}(\mathrm{pfx}(L), f)$. As the map $\mathbf{s}$ is one-on-one, all other claims follow from the same argument.

Given Lem. 2 and referring to the automaton realisation G of L and E from above, Problem 1 can be equivalently stated as the following control strategy synthesis problem over a two-player game.

Problem 3. Given an automaton G and two sets of marked states $F_L, F_E \subseteq Q$ s.t. $F_E \subseteq F_L$, synthesize a control strategy $h : q_{\mathrm{init}}Q^* \to \Gamma$ s.t.

$$\mathbf{L}(G, h) \cap \mathbf{L}_m(G, F_L) \subseteq \mathbf{L}_m(G, F_E), \text{ and} \tag{3a}$$

$$\mathbf{L}(G, h) = \mathrm{pfx}(\mathbf{L}(G, h) \cap \mathbf{L}_m(G, F_L)), \tag{3b}$$

or determine, that no such strategy exists.

Intuitively, such a control strategy enforces that if a marked state is attained by the plant, it must be a state marked by the specification, Eq. (3a), and it is always possible for the plant to organize its moves s.t. it will eventually reach one of its marked states, Eq. (3b), and thereby generating a word in E.

3.2 Appropriate Winning Conditions for Supervision

Given the two player game outlined above, reactive synthesis is typically concerned with synthesizing h s.t. all traces in $\mathbf{L}(G, h)$ fulfill a given *winning condition* φ. In the context of regular *-languages, winning conditions can be interpreted as another regular language $M_\varphi \subseteq \Sigma^*$. This would amount to synthesizing h s.t. $\mathbf{L}(G, h) \subseteq M_\varphi$. However, if we look back at Problem 3, the requirements stated in (3) are of a slightly different form. We therefore cannot use an existing game-solving algorithm "of-the-shelf". Below we outline, how we can still derive a fixed-point algorithm to compute a control strategy solving Problem 3.

A Reachability Game. An obvious first attempt to solve the proposed supervisor synthesis problem via the outlined two-player game is to define $M_\varphi := \mathbf{L}_m(G, F_E) = E$. This requires that states marked by F_E must eventually be reached by all traces over G consistent with h. This defines a reachability game over G. It is well known, see e.g., Baier and Katoen (2008), that all states from which the controller can *enforce* that F_E is eventually reached, can be computed by the fixed-point

$$\mu X \,.\, \mathrm{Pre}_0(X) \cup F_E, \tag{4}$$

where $X \subseteq Q$. Here, Pre_0 defines the *controllable predecessor operator* which returns all states from which player 0 can provide a control pattern γ which ensures that, no matter which $\sigma \in \gamma$ the plant chooses, it will always end up in the states $X \subseteq Q$. As $\Sigma_{uc} \subseteq \gamma$ by definition, the operator Pre_0 is defined for the particular two-player game under consideration by

$$\mathrm{Pre}_0(X) := \{q \in Q | \delta(q, \Sigma) \cap X \neq \emptyset \wedge \delta(q, \Sigma_{uc}) \subseteq X\}\,. \tag{5}$$

As an example for its application, consider the automaton G depicted in Fig. 1 (bottom left) and consider $X = Q_\mathrm{m} = F_E$. Now one can verify that $q_1 \notin \mathrm{Pre}_0(Q_\mathrm{m})$ as there exists an uncontrollable transition from q_1 to q_2 which does not lead to Q_m. A similar reasoning shows that also $q_7 \notin \mathrm{Pre}_0(Q_\mathrm{m})$. Intuitively, the controller cannot *enforce* the plant to make progress towards its marked states when in q_1 or q_7. Because of this, the reachability fixed-point, Eq. (4), terminates with only q_3 and q_4 as winning states. This, however, is not the set we are looking for (as indicated by G_{cl} in Fig. 1 (bottom right)).

A Cooperative Reachability Game. In the context of the proposed supervisor synthesis problem, Problem 3, we know that the plant, if allowed to do so, will eventually transition to its markings by itself, without the supervisor enforcing this. Hence, we know that there exists no trace of G_L consistent with h which will loop between q_1 and q_2 or in q_7 forever. This is the reason why (3a) in Problem 3 requires the weaker inclusion $\mathbf{L}(G, h) \cap L \subseteq \mathbf{L}_m(G, F_E)$ instead of $\mathbf{L}(G, h) \subseteq \mathbf{L}_m(G, F_E)$. However, to exclude trivial solutions to this requirement, i.e., the supervisor always blocking the plant before reaching any marking, resulting in $\mathbf{L}(G, h) = \{\epsilon\}$ and thereby trivially satisfying (3a), Problem 3 additionally imposes the non-conflictingness requirment, Eq. (3b). That is, in any instance of the play the plant must be able to play such that it will eventually attain some marked state.

It was shown by Majumdar et al. (2019), that for strategy synthesis w.r.t. a particular fragment of linear temporal logic (LTL) specifications, namely GR(1) specifications (Bloem et al., 2012), the non-conflictingness requirement, Eq. (3b), can be incorporated by substituting the player-0 controllable predecessor Pre_0 in the original fixed-point algorithm by a cooperative one. Transferring this idea to reachability games with the particular SCT inspired interpretation of player 0, we obtain the cooperative reachability fixed-point

$$\nu Y \,.\, \mu X \,.\, \mathrm{CondPre}(X, Y) \cup F_E \tag{6}$$

where

$$\begin{aligned}&\mathrm{CondPre}(X, Y) := \\ &\quad \{q \in Q | \delta(q, \Sigma) \cap X \neq \emptyset \wedge \delta(q, \Sigma_{uc}) \in X \cup Y\}\,.\end{aligned} \tag{7}$$

Let us now apply (6) to the automaton G in Fig. 1 (bottom left). First, one should note that the outer fixed-point over Y is initialized by the full state set Q, while the inner fixed-point over X is initialized by the empty state set. Let us now fix $Y^0 = Q$ and consider the step-by-step evolution of the inner fixed-point. We have $X^0 = \emptyset$ and $X^1 = F_E = \{q_3, q_4\}$. Now we see that q_1 will indeed be added to the set of winning states in the second iteration, that is $q_1 \in \mathrm{CondPre}(F_E, Q)$. One can therefore verify that the inner fixed-point terminates with $X^\infty = Q \setminus \{q_5, q_8\}$, as q_5 and q_8 are the only states that do not have a path to $Q_\mathrm{m} = F_E$. With this, we update Y to $Y^1 := Q \setminus \{q_5, q_8\}$ and re-start the inner fixed-point computation. As $q_8 \notin Y^1$, this iteration now also removes p_6 from the winning state set. Updating Y again to $Y^2 := Q \setminus \{q_5, q_6, q_8\}$ finally also removes q_9 and the fixed-point algorithm terminates with $Y^\infty = \{q_0, \ldots, q_4, q_7\}$.

Interestingly, we have $q_4 \in Y^\infty$, i.e., the fixed-point in (6) determines q_4 to be winning. This is not surprising, as (6) computes the set of states which allow the plant and the controller to cooperate s.t. a marked state will be reached "at least once". Thereby the algorithm does not care about the behavior of the closed loop after this happened. As control patterns Γ are defined such that they always enable all uncontrollable transitions, the closed loop behavior will however still allow the plant to move to q_5 and by this cause a live-lock of the controlled system which is obviously unintended. Hence, (3a) will be violated.

A Coorperative Büchi-Type Game. To circumvent the above problem with the cooperative reachability game, we need to make sure that whenever the plant is able to leave an attained marking, a marking must be re-reachable. This type of game, where marked states need to be reached, are called Büchi games. It is well known, see e.g. (Baier and Katoen, 2008), that the fixed-point algorithm for solving Büchi games is given by

$$\nu Y \,.\, \mu X \,.\, \mathrm{Pre}_0(X) \cup (\mathrm{Pre}_0(Y) \cap F_E)\,. \tag{8}$$

Following Majumdar et al. (2019), its cooperative version is given by substituting the inner-most controllable predecessor by the cooperative one, leading to

$$\nu Y \,.\, \mu X \,.\, \mathrm{CondPre}(X, Y) \cup (\mathrm{Pre}_0(Y) \cap F_E)\,. \tag{9}$$

Intuitively, the additional controllable pre-operator $\mathrm{Pre}_0(Y)$ conjoined with F_E ensures that in states $q \in F_E$ the supervisor can *enforce* the plant to continue operating in a manner that allows to re-reach a marked state.

Unfortunately, this requirement turns out too strong in the context of Problem 3. To see this, let us consider a slightly modified version of G in Fig. 1 where the only outgoing transition of q_3 is the controllable one reaching q_4. I.e., q_3 cannot transition to

q_1 via the un-controllable transition indicated in Fig. 1. Now one can verify that for the first iteration of the inner fixed-point of (8) with $Y^0 = Q$, we again obtain $X^\infty = Q \setminus \{q_5, q_8\}$ and therefore update Y to $Y^1 = Q \setminus \{q_5, q_8\}$. Now one can verify that $q_4 \notin \mathrm{Pre}_0(Y^1)$. With this, the next iteration of the inner fixed-point terminates with $X^\infty := Q \setminus \{q_4, q_5, q_6, q_8\}$ resulting in the update $Y^2 := Q \setminus \{q_4, q_5, q_6, q_8\}$. With this, however, we further have $q_3 \notin \mathrm{Pre}_0(Y^2)$ (as we assume the transition to q_1 to be non-existent). Hence, the fixed-point removes q_3 and terminates with $Y^\infty = \emptyset$.

In the context of SCT w.r.t. $*$-languages, however, there is no need to remove q_3 from the set of winning states. For correctness we only need to make sure that uncontrollable transitions starting in marked states only reach states that allow to re-reach a marked one but we do not need to *enforce* progress out of marked states. We obtain this, by weakening the controllable predecessor operator $\mathrm{Pre}_0(Y)$ into the universal predecessor operator $\mathrm{Pre}_\forall(Y)$, defined by

$$\mathrm{Pre}_\forall(X) := \{q \in Q | \delta(q, \Sigma_{uc}) \subseteq X\}. \tag{10}$$

With this substitution, we obtain the fixed-point

$$\nu Y . \mu X . \mathrm{CondPre}(X, Y) \cup (\mathrm{Pre}_\forall(Y) \cap F_E). \tag{11}$$

While we will show shortly that this is indeed the right fixed-point to compute a maximally permissive supervisor solving Problem 3, we first slightly simplify (11). For this, we define the existential predecessor operator

$$\mathrm{Pre}_\exists(X) := \{q \in Q | \delta(q, \Sigma) \cap X \neq \emptyset\}. \tag{12}$$

Now, it follows from the monotonicity of the fixed-point in (11) that we always have $X \subseteq Y$ for any iteration of (11). With this, one can verify that $\mathrm{CondPre}(X, Y) = \mathrm{Pre}_\exists(X) \cap \mathrm{Pre}_\forall(Y)$. Using this substitution and applying common de-Morgan laws, we obtain the simplified fixed-point

$$W := \nu Y . \mu X . (\mathrm{Pre}_\exists(X) \cup F_E) \cap \mathrm{Pre}_\forall(Y). \tag{13}$$

In the remainder of this section we show that (13) indeed computes the desired set of winning states, and thereby, allows us to extract the desired maximally permissive supervisor if and only if $q_{init} \in W$, hence solving Problem 3.

3.3 *Correctness of Strategy Synthesis*

First, consider the set $W \subseteq Q$ computed via (13) and observe that for

$$h(q) := \begin{cases} \Sigma_{uc} \cup \{\sigma \in \Sigma \mid \delta(q, \sigma) \in W\}, & q \in W \\ \Sigma, & \text{otherwise} \end{cases} \tag{14}$$

it holds that $h(q) \subseteq \Sigma_{uc}$. Therefore, h is obviously a memoryless control strategy over G.

Indeed, we can show that from any state $q \in W$ the strategy h enforces that the plant will evolve s.t. (3) holds. This establishes soundness of (13) as summarized in the following lemma and proven in the appendix.

Lemma 4. Given G as in Problem 3, $W \subseteq Q$ computed via (13) and $h : Q \to \Gamma$ as in (14), it holds for all $q \in W$ that [1]

$$\mathbf{L}_q(G, h) \cap \mathbf{L}_{m,q}(G, F_L) \subseteq \mathbf{L}_{m,q}(G, F_E), \text{ and} \tag{15a}$$
$$\mathbf{L}_q(G, h) = \mathrm{pfx}\,(\mathbf{L}_q(G, h) \cap \mathbf{L}_{m,q}(G, F_L)). \tag{15b}$$

This shows that we can extract a memoryless control strategy from W to solve Problem 3 if $q_{\mathrm{init}} \in W$, as for $q = q_{\mathrm{init}}$ equation (15) coincides with (3). To fulfill the second requirement of Problem 3, i.e., to surely determine that no controller exists, we also need to prove completeness of (13). That is, we show that for all $q \in Q \setminus W$ there exists no control strategy $h : q_{init}Q^* \to \Gamma$ s.t. (3) holds. This is formalized in the following lemma and proven in the appendix.

Lemma 5. Given G as in Problem 3 and $W \subseteq Q$ computed via (13), it holds for all $q \in \overline{W} = Q \setminus W$ and any $h : q_{\mathrm{init}}Q^* \to \Gamma$ that either

$$\mathbf{L}_q(G, h) \cap \mathbf{L}_{m,q}(G, F_L) \not\subseteq \mathbf{L}_{m,q}(G, F_E), \text{ or} \tag{16a}$$
$$\mathbf{L}_q(G, h) \not\subseteq \mathrm{pfx}\,(\mathbf{L}_q(G, h) \cap \mathbf{L}_{m,q}(G, F_L)). \tag{16b}$$

With this, it immediately follows that there exists a control strategy s.t. (3) holds, if and only if $q_{\mathrm{init}} \in W$. As it turns out, the strategy h defined in (14) is maximally permissive in the folllowing sense:

Proposition 6. Given the premises of Lem. 4 s.t. $q_{init} \in W$, the memoryless strategy h defined in (14) is maximally permissive w.r.t. (3), that is, for all $h' : q_{\mathrm{init}}Q^* \to \Gamma$ s.t. (3) holds, we have that $\mathbf{L}(G, h') \subseteq \mathbf{L}(G, h)$.

Proof. Pick h' s.t. (3) holds and assume, for the purpose of a contradiction, that there exists $s = \sigma_0 \ldots \sigma_k \in \mathbf{L}(G, h')$ s.t. $s \notin \mathbf{L}(G, h)$. Let $\pi = \mathbf{s}^{-1}(s) = q_{init}\sigma_0 \ldots \sigma_k q_{k+1} \in \mathbf{P}(G, h')$ and observe that $\pi \notin \mathbf{P}(G, h)$. This implies that there exists $i \in [0; k]$ s.t. $\sigma_i \in h'(q_0 \ldots q_i)$ but $\sigma_i \notin h(q_i)$. Now recall from (14) that this implies that $\delta(q_i, \sigma_i) \notin W$ (as otherwise $\sigma_i \in h(q_i)$). With this we have $q_{i+1} \in \overline{W}$. With this, it follows from Lem. 5 that there exists no $h'' : q_{i+1}Q^* \to \Gamma$ s.t. (3) holds. This immediately implies that (3) cannot hold for h' which concludes the contradiction.

This concludes our evaluation. Combining Lem. 2, Lem. 4, Lem. 5 and Prop. 6, we have the following corollary to summarize the developments in this paper.

Corollary 7. Given a plant L and a relatively closed specification $\emptyset \neq E \subseteq L$, let $G = (Q, \Sigma, \delta, q_{\mathrm{init}}, Q_{\mathrm{m}})$ be an automaton with two sets of marked states $F_E \subseteq F_L \subseteq Q$ s.t. $\mathbf{L}_m(G, F_L) = L$, $\mathbf{L}_m(G, F_E) = E$, and $\mathbf{L}(G) = \mathrm{pfx}\,(L)$. Further, let $W \subseteq Q$ be computed via (13) and $h : Q \to \Gamma$ as in (14). If and only if $q_{\mathrm{init}} \in W$, there exists a supervisor $f : \mathrm{pfx}\,(L) \to \Gamma$ s.t. (2) holds. In particular, if $q_{\mathrm{init}} \in W$ the supervisor $f : h \circ \mathbf{s}^{-1}$ fulfills (2) and is maximally permissive.

4. CONCLUSION

In this paper, we have revisited the common problem of supervisory control, Problem 1, as originally proposed by Ramadge and Wonham (1987). We have transformed this problem to a strategy synthesis problem over a two-player game, Problem 3, as it is common in the field of reactive synthesis, and we provide an algorithmic solution via the fixpoint (13). As our main result, summarized in Cor. 7, the closed-loop behaviour achieved by our supervisor is maximally permissive and, hence, in this regard matches the literature (Ramadge and Wonham, 1987; Wonham and Ramadge, 1987). This is an interesting observation, as not only the perspectives taken but also the subsequent technical development differs significantly. To this end, our result contributes to the ongoing discussion on the relationship between reactive synthesis and supervisory control; e.g., Ehlers et al. (2017); Schmuck et al. (2020).

Despite the match in the resulting closed-loop behaviour, the supervisors synthesised by our approach are not identical to

[1] Given a state $q \in Q$ we denote by $\mathbf{L}_q(G, \cdot)$ and $\mathbf{L}_{m,q}(G, \cdot)$ the respective language obtained when interpreting q as the initial state of G.

those obtained by the established procedures. For this observation, consider a string outside the prefix of the supremal closed-loop behaviour. In the classical setting, the supervisor may map this string to any control pattern, trusting in the fact that such a string will not occur in the closed loop. Hence, if for whatever reason the actual plant deviates from the design model the supervisor makes no attempt to faithfully operate the plant. In this regard, the supervisors synthesised by our approach perform better in that they apply a well chosen control pattern as long as the design model is within a winning state. If, after a deviation from the nominal behaviour by chance a winning state is again attained, the supervisor will "catch on" to win its play. We envisage this to be of practical relevance in the context of fault-tolerant control, control re-configuration and re-initialisation after break down.

Appendix A. PROOFS

Proof. [of Lem. 4] First, let us consider one entire run of the inner μ-iteration of (13) at a stage where the outer ν-iteration has attained its fixpoint $Y^\infty = W$. I.e., we consider the monotone sequence $(X_i)_{i\in\mathbb{N}_0}$ generated by

$$X_0 := \emptyset, \quad X_{i+1} := X_i \cup ((\mathrm{Pre}_\exists(X_i) \cup F_E) \cap \mathrm{Pre}_\forall(W)). \quad (A.1)$$

In particular, we have that $W = Y^\infty = X^\infty = \cup\{X_i \mid i \in \mathbb{N}_0\}$ and $W \subseteq \mathrm{Pre}_\forall(W)$. With this, one can verify that

$$\forall q \in W, \sigma \in h(q) \,.\, \delta(q,\sigma)! \Rightarrow \delta(q,\sigma) \in W. \quad (A.2)$$

For $\sigma \in \Sigma_\mathrm{c}$ the claim in (A.2) follows directly from the definition of h, Eq. (14). For $\sigma \in \Sigma_\mathrm{uc}$, the claim in (A.2) follows from the fact that $W \subseteq \mathrm{Pre}_\forall(W)$ and the definition of $\mathrm{Pre}_\forall(W)$, Eq. (10), implies that $\delta(q,\sigma) \in W$ for all enabled uncontrollable transitions.

Now let us introduce the following ranking of states $q \in W$:

$$\mathrm{rank}(q) := i \in \mathbb{N}_0 \text{ iff } q \in X_i \setminus X_{i-1}. \quad (A.3)$$

Note that $\mathrm{rank}(q) \geq 1$ for all $q \in W$ and $\mathrm{rank}(q) = 1$ iff $q \in F_E \cap W$ since $X_0 = \emptyset$ and $\mathrm{Pre}_\exists(\emptyset) = \emptyset$. With this, we show that

$$\forall q \in W \,.\, \mathbf{L}_q(G,h) \cap \mathbf{L}_{m,q}(G,F_E) \neq \emptyset. \quad (A.4)$$

We show this claim by constructing a play $\pi = s_0\sigma_0 s_1\sigma_1 \ldots s_k \in \mathbf{P}_q(G,h)$ with $s_0 = q$ generating the string $\nu = \sigma_0\sigma_1\ldots\sigma_{k-1} \in \mathbf{L}_q(G,h) \cap \mathbf{L}_{m,q}(G,F_E)$. As $s_0 = q \in W$ we know $\mathrm{rank}(s_0) = i$ is defined. If $i = 1$ we know $s_0 \in F_E$ and therefore $\epsilon \in \mathbf{L}_q(G,h)$ and $\epsilon \in \mathbf{L}_{m,q}(G,F_E)$. Now assume $i > 1$. Then $s_0 \in X_i$ and we know from the definition of $\mathrm{Pre}_\exists$, Eq. (12), and the definition of h, Eq (14), that there exists $\sigma_0 \in h(s_0)$ s.t. $s_1 = \delta(s_0,\sigma_0) \in X_{i-1}$. By iteratively applying this argument we reach a state $s_k \in X_1 = F_E$. As the constructed sequence is compliant with h and a run on G ending in F_E, we have $\nu \in \mathbf{L}_q(G,h) \cap \mathbf{L}_{m,q}(G,F_E)$.

As $F_E \subseteq F_L$ one can verify that (A.4) directly implies (3b). First, recall that $\supseteq$ always holds and we are left with showing $\subseteq$. For this, we pick a play $\pi = s_0\sigma_0 s_1\sigma_1\ldots s_k \in \mathbf{P}_q(G,h)$ with $s_0 = q \in W$ generating the string $\nu = \sigma_0\sigma_1\ldots\sigma_{k-1} \in \mathbf{L}_q(G,h)$. Then we know from inductively applying (A.2), that $s_n \in W$. Therefore we can use (A.4) to pick a sequence β s.t. $\nu\beta \in \mathbf{L}_q(G,h) \cap \mathbf{L}_{m,q}(G,F_E)$. As $F_E \subseteq F_L$, and therefore $\mathbf{L}_{m,q}(G,F_E) \subseteq \mathbf{L}_{m,q}(G,F_L)$ this immediately implies that $\nu \in \mathrm{pfx}\,(\mathbf{L}_q(G,h) \cap \mathbf{L}_{m,q}(G,F_L))$.

Now it remains to show that (3a) also holds. Towards this goal, we first show that

$$q \in W \cap F_L \Rightarrow q \in F_E. \quad (A.5)$$

As $F_E \subseteq F_L$, the claim immediately holds if $q \in W \cap F_E$. For the sake of a contradiction, let us assume that there exists a state $q \in W \cap F_L \setminus F_E$. As E is relatively closed w.r.t. L we know that there exists some $p \in P$ (of G_L) s.t. $q = (p, s_\perp)$. Further, we have that $\delta'_E(s_\perp, \Sigma^*) = s_\perp$ and $s_\perp \notin S_m$ (of G'_E). As $G = G_L \times G'_E$, this implies $\mathbf{L}_q(G,h) \cap \mathbf{L}_{m,q}(G,F_E) = \emptyset$. Using (A.4) this implies $q \notin W$, which concludes the contradiction.

Given (A.5), one can verify that (3a) holds. For this, pick any $\pi = s_0\sigma_0 s_1\sigma_1\ldots s_k \in \mathbf{P}_q(G,h)$ where $s_0 = q \in W$ s.t. $\nu = \sigma_0\sigma_1\ldots\sigma_{k-1} \in \mathbf{L}_q(G,h) \cap \mathbf{L}_{m,q}(G,F_L)$. Then we know from inductively applying (A.2), that $s_n \in W \cap F_L$, and therefore from (A.5) that $s_n \in F_E$. Hence, $\nu \in \mathbf{L}_{m,q}(G,F_E)$.

Proof. [of Lem. 5] We first negate the fixed-point in (13) to formally characterize the set of states $\overline{W} = Q \setminus W$. Using the negation rule of the μ-calculus, i.e., $\neg(\mu X \,.\, F(X)) = \nu X \,.\, \neg F(\neg X)$, and common de-Morgan laws, we obtain

$$\overline{W} = \mu Y \,.\, \nu X \,.\, \left(Q \setminus F_E \cap \overline{\mathrm{Pre}}_\exists(X)\right) \cup \overline{\mathrm{Pre}}_\forall(Y) \quad (A.6)$$

with

$$\overline{\mathrm{Pre}}_\exists(X) = \{q \in Q \mid \delta(q,\Sigma) \subseteq X\} \quad (A.7)$$

$$\overline{\mathrm{Pre}}_\forall(Y) = \{q \in Q \mid \delta(q,\Sigma_{uc}) \cap Y \neq \emptyset\} \quad (A.8)$$

First, let us consider the last run over the outer μ-iteration before termination. In this case we have $Y_0 = \emptyset$ and denote by Y_i for $i > 0$ the set obtained from the i-th iteration of the outer fixed-point. For $i \geq 1$ we have $X_i^\infty = Y_i$ as the value of the inner fixed-point over X that computes the i-th iteration of Y. Then we obtain the monotone sequence $(Y_i)_{i\in\mathbb{N}_0}$ which satisfies

$$Y_0 := \emptyset, \quad Y_i = Y_{i-1} \cup \left(Q \setminus F_E \cap \overline{\mathrm{Pre}}_\exists(Y_i)\right) \cup \overline{\mathrm{Pre}}_\forall(Y_{i-1}). \quad (A.9)$$

In particular, we have that $\overline{W} = X_\infty^\infty = Y^\infty = \cup\{Y_i \mid i \in \mathbb{N}_0\}$. Now let us introduce the following ranking of states $q \in \overline{W}$:

$$\overline{\mathrm{rank}}(q) := i \in \mathbb{N}_0 \text{ iff } q \in Y_i \setminus Y_{i-1}. \quad (A.10)$$

Note that $\overline{\mathrm{rank}}(q) \geq 1$ for all $q \in \overline{W}$ since $Y_0 = \emptyset$, With this, we can verify that for any state $q \in \overline{W}$ with $\overline{\mathrm{rank}}(q) = i$ we have that either (a) $q \notin F_E$ and $q' = \delta(q,\sigma) \in Y_i$ i.e., $\overline{\mathrm{rank}}(q') \leq i$ for all σ enabled in q, or (b) there exists $\sigma \in \Sigma_\mathrm{uc}$ which is enabled in q and $q' = \delta(q,\sigma) \in Y_{i-1}$, i.e., $\overline{\mathrm{rank}}(q') \leq i - 1$.

Now consider a state $q \in \overline{W}$ and any control strategy $h : q_\mathrm{init}(Q)^* \to \Gamma$. Let us first assume that there exists no $\sigma \in \Sigma$ s.t. $(q,\sigma)!$. This implies that case (a) holds for q, i.e., $q \notin F_E$. Now we can either have $q \in F_L$ in which case (16a) is true, or $q \notin F_L$, in which case (16b) is true, as no outgoing transitions of q implies that no state $q' \notin F_L$ is reachable from q. I.e., if q has no enabled transitions, (16) holds.

Now let us assume that there exists an enabled uncontrollable transition $\sigma \in \Sigma_{uc}$ in q s.t. $(q,\sigma)!$. If in addition (a) holds, then this transition can always be taken by the plant no matter which control pattern γ the strategy h chooses for q and will always result in a state $q' \in \overline{W}$ with equal or reduced rank. I.e., if $\overline{\mathrm{rank}}(q) = i$ then $\overline{\mathrm{rank}}(q') \leq i$. On the other hand, if in addition (b) holds, then we know that there exists a possibly different $\sigma' \in \Sigma_{uc}$ which is always enabled no matter which control pattern h chooses for q and allows the plant to transition to a state $q' \in \overline{W}$ while reducing the rank.

Now let us assume that there exists no enabled uncontrollable transition $\sigma \in \Sigma_{uc}$ in q s.t. $(q,\sigma)!$. This implies that (a) must hold. Then we know that all enabled controllable transitions σ'

lead to a state $q' \in \overline{W}$ with equal or reduced rank. Now we have two cases. If at least one such σ' is contained in γ chosen by h in q, we know that the plant can take this transition to $q' \in \overline{W}$ while not increasing the rank. If this is not the case, we observe from (a) that $q \notin F_E$. It then follows from the same reasoning as in the blocking scenario above that (16b) is true.

The above reasoning shows that either we encounter a blocking situation (due to disabled events, or due to a control strategy h disabling all enabled controllable events) and thereby verify that (16) holds, or any control strategy h allows the plant to proceed to a state in $\overline{W}$ while never increasing the rank. To show that (16) also holds in the second case, let us iteratively construct a tree T rooted in q which collects all traces over G generated by the plant which ensure it is staying in $\overline{W}$. Further, let us label each node of T by either (a) or (b), depending on which case applies, and observe that for all nodes q' of the tree we have $q' \in \overline{W}$. Further, we label a node by $(\perp)$ if it is blocking due to one of the two previously discussed scenarios.

First, let us assume that there exists a node q'' in T with label $(\perp)$ which is reached from q with trace $\beta = q\sigma_0 \ldots q''$ generating the event sequence $\beta' \in \mathbf{L}_q(G, h)$. As $q'' \in \overline{W}$ we can apply the same reasoning as above to determine that (16) holds. Now, consider the case that there exists no node q'' in T with label $(\perp)$ which is reachable from q, implying that T is an infinite tree. Now assume by contradiction that case (b) occurs infinitely often in T. By König's lemma, it follows that there is a path π in T along which (b) occurs infinitely often. However, whenever (b), occurs, the rank of the node decreases. Further, the occurence of (a) can never increase the rank. As the rank is finite, this shows that (b) can only happen finitely often along each trace of T. This reasoning implies that along every branch of T (enumerated by $k \in \mathbb{N}$) there exists a *finite* prefix $s_k \in \text{pfx}(\mathbf{L}_q(G, h))$ leading to a state q_k at which a subtree T'_k is rooted in which only case (a) occurs. As case (a) renders $\overline{W}$ closed under any enabled transition, we know that no state F_E is ever reachable along these traces (as nodes for which case (a) applies are by definition not in F_E). This implies that $s_k \notin \text{pfx}(\mathbf{L}_{m,q}(G, F_E))$, which shows that (16b) is true.

REFERENCES

Aucher, G. (2014). Supervisory control theory in epistemic temporal logic. In *AAMAS '14*, 333–340.

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

Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., and Sahar, Y. (2012). Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3), 911 – 938.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, second edition.

Cassez, F. and Tripakis, S. (2008). Fault diagnosis with static and dynamic observers. *Fundamenta Informaticae*, 88(4), 497–540.

Ehlers, R., Lafortune, S., Tripakis, S., and Vardi, M.Y. (2017). Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems*, 27(2), 209–260.

Finkbeiner, B. (2016). Synthesis of reactive systems. Technical report, Universität des Saarlandes.

Hélouët, L., Marchand, H., and Ricker, L. (2018). Opacity with powerful attackers. *WODES'18*, 51(7), 464 – 471.

Jiang, S. and Kumar, R. (2006). Supervisory control of discrete event systems with ctl* temporal logic specifications. *SIAM Journal on Control and Optimization*, 44(6), 2079–2103.

Lafortune, S. (2019). Discrete event systems: Modeling, observation, and control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2, 141–159.

Majumdar, R., Piterman, N., and Schmuck, A.K. (2019). Environmentally-friendly gr (1) synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 229–246. Springer.

Maler, O. (2002). Control from computer science. *Annual Reviews in Control*, 26(2), 175 – 187.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25, 206–230.

Ramezani, Z., Krook, J., Fei, Z., Fabian, M., and Akesson, K. (2019). Comparative case studies of reactive synthesis and supervisory control. In *2019 18th European Control Conference (ECC)*, 1752–1759. IEEE.

Sakakibara, A. and Ushio, T. (2018). Hierarchical control of concurrent discrete event systems with linear temporal logic specifications. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E101.A(2), 313–321. doi:10.1587/transfun.E101.A.313.

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

van Hulst, A.C., Reniers, M.A., and Fokkink, W.J. (2017). Maximally permissive controlled system synthesis for non-determinism and modal logic. *Discrete Event Dynamic Systems*, 27(1), 109–142.

Wonham, W.M. and Cai, K. (2019). *Supervisory control of discrete-event systems*. Communications and Control Engineering. Springer.

Wonham, W.M. and Ramadge, P.J. (1987). On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3), 637–659.

Wonham, W., Cai, K., and Rudie, K. (2018). Supervisory control of discrete-event systems: A brief history. *Annual Reviews in Control*.

Yin, X. and Lafortune, S. (2016a). Synthesis of maximally permissive supervisors for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 61(5), 1239–1254.

Yin, X. and Lafortune, S. (2016b). A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 61(8), 2140–2154.

Yin, X. and Lafortune, S. (2019). A general approach for optimizing dynamic sensor activation for discrete event systems. *Automatica*, 105, 376 – 383.