On the Relation between Reactive Synthesis and Supervisory Control of Non-Terminating Processes

Anne-Kathrin Schmuck[*], Thomas Moor[†], and Rupak Majumdar[*]

Abstract. *Reactive synthesis* and *supervisory control theory* both provide a design methodology for the automatic and algorithmic design of digital systems from declarative specifications. The *reactive synthesis* approach originates in computer science, and seeks to synthesise a system that interacts with its environment over time and that, doing so, satisfies a prescribed specification. Here, the distinguishing feature when compared to other synthesis problems in computer science is that the interaction is temporal in that it explicitly refers to a sequence of computation cycles. *Supervisory control* originates in control theory and seeks to synthesise a controller that – in closed-loop configuration with a plant – enforces a prescribed specification over time. The distinguishing feature compared to other branches of control is that all dynamics are driven by discrete events as opposed to continuous physical signals.

While both methods apparently are closely related, the technical details differ significantly. We provide a formal comparison which allows us to identify conditions under which one can solve one synthesis problem using methods from the other one; we also discuss how the resulting solutions compare. To facilitate this comparison, we give a unified introduction to reactive synthesis and supervisory control and derive formal problem statements and a characterisation of their solutions in terms of ω-languages. Recent contributions to the two fields address different aspects of the respective problem, and we expect the formal relationship identified in this paper to be useful in that it allows the application of algorithmic techniques from one field in the other.

This report is published to provide additional propositions, proofs and illustrative examples to support a paper currently under review for WODES'18.[1]

[*]Max Planck Institute for Software Systems, Kaiserlautern, Germany
[†]Friedrich-Alexander Universität Erlangen-Nürnberg, Germany
[1]Revision December 19, 2017

Contents

Introduction

Reactive synthesis (RS) addresses the systematic design of digital systems that dynamically interact with their environment by alternating input readings from discrete variables and output assignments to discrete variables. The progress of time is modelled by successive computation cycles and the current output assignment is considered to depend on all past input readings. Thus, the digital system imposes a causal feedback on the environment and it is therefore referred to as a *reactive module*. In practical applications, a reactive module is realised as a finite automaton. The synthesis problem here is to construct a reactive module realized by a finite automaton that exhibits a behaviour that satisfies some prescribed specification. The problem of reactive synthesis was first proposed by [Church, 1957] with solutions provided by [Büchi and Landweber, 1969], and [Rabin, 1972]. It is since then an active field of research in computer science, addressing temporal logic specifications (e.g. [Pnueli and Rosner, 1989, Emerson and Jutla, 1991, Maler et al., 1995, Thomas, 1995]), partial observation (e.g. [Kupferman and Vardi, 2000]), stochasticity (e.g. [de Alfaro and Henzinger, 2000, de Alfaro et al., 2007, Chatterjee and Henzinger, 2012]), and, most relevant for the present report, environment assumptions (see [Bloem et al., 2014] and [Brenguier et al., 2017] for an overview). For a comprehensive introduction to the field see e.g. [Thomas, 1995, Finkbeiner, 2016].

Supervisory Control Theory (SCT) is a branch of control theory that also addresses the systematic design of digital systems. It models systems as *discrete-event systems* — dynamical systems in which relevant variables are of finite range and in which changes of their respective values are referred to as *events*. The synthesis problem in control theory is to construct a controller that provides causal feedback to a given plant such that the closed-loop system satisfies a prescribed specification. For supervisory control, the plant is a discrete-event system and the causal feedback, referred to as the *supervisor*, maps the past event sequence to a control pattern in order to restrict the plant behaviour. Supervisory control theory was originally proposed by [Ramadge and Wonham, 1987] and now is an established field of research. Topics addressed include partial observation (e.g. [Lin and Wonham, 1988, Cai et al., 2015, Yin and Lafortune, 2016]), robustness (e.g. [Cury and Krogh, 1999, Bourdon et al., 2005]), modularity (e.g. [Ramadge and Wonham, 1989, de Querioz and Cury, 2000]), hierarchical control architectures (e.g. [Zhong and Wonham, 1990, Wong and Wonham, 1996, Schmidt et al., 2008]), fault-tolerance (e.g. [Wen et al., 2008, Paoli et al., 2011, Moor, 2016]), and, most relevant for this report, infinite behaviours (e.g. [Ramadge, 1989, Thistle and Wonham, 1994b]).

Both design methodologies seek to synthesize a causal feedback map that operates on a finite alphabet and that satisfies a formal specification. When used to synthesize solutions for particular instances of a given synthesis problem, both techniques appear to be closely related. However, the technical details differ significantly. In this paper we provide a formal comparison, allowing us to identify conditions under which one can solve one synthesis problem via the respective other one and we discuss how the resulting solutions compare. To facilitate this comparison, we give a concise introduction to RS and SCT and derive formal problem statements and a characterisation of their solutions in terms of ω-languages. Recent contributions to the two fields focus attention on different aspects of the respective problem, and we expect the formal relationship identified in this paper to be useful in that it allows the application of algorithmic techniques from one field to the other.

Scope Regarding *reactive synthesis*, our study considers a variant that explicitly addresses assumptions on the environment behaviour, as these assumptions correspond to the prescribed plant behaviour in supervisory control. For ease of comparison, we consider both the assumption and the specification to be given abstractly as ω-languages —formal languages of *infinite* words— and for algorithmic effectiveness, as ω-regular languages which allow automata-based representations. For ease of illustration, we restrict attention to specifications given as *deterministic* Büchi automata. While deterministic Büchi automata capture only a strict subset of ω-regular languages, it allows us to keep the notation and algorithms simple; our comparisons can be extended to the full class of all ω-regular specifications. While many papers on RS use specifications given in linear temporal logic (LTL) [Pnueli, 1977], it is well understood how such formulae can be translated into ω-automata [Vardi and Wolper, 1986, Safra, 1988].

Regarding *supervisory control*, most of the literature, including [Ramadge and Wonham, 1987], refers to $*$-languages as their base model, i.e., formal languages of *finite* words. In this setting, synthesis can enforce safety properties while maintaining liveness properties present in the plant behaviour. This contrasts the design of reactive modules where the synthesis of liveness properties is conceived a relevant challenge. We therefore conduct our study for a branch of supervisory control that addresses the synthesis problem for ω-languages; see [Ramadge, 1989, Thistle and Wonham, 1994b, Thistle, 1995], where the authors explicitly relate their work to Church's problem. For a comprehensive introduction to SCT for ω-languages see also [Moor, 2017].

Contribution Within this perspective of our choice, our contribution is threefold:

(I) We show that one can solve the considered RS problem using SCT to obtain a reactive module which will not falsify the assumptions on the environment.

(II) We show that one can solve the considered synthesis problem from SCT using RS for restricted subclasses of plant (resp. environment) behaviours.

(III) We establish equivalence of the two synthesis problems regarding solvability for the subclasses considered in (II).

The considered RS problem is formalised by an implication style logic formula, i.e., the specification is such that if the assumptions are satisfied, then a guarantee shall be provided. Hence, a valid solution to the synthesis problem might *falsify the assumption*. SCT seeks to avoid this issue by requiring that valid solutions to the synthesis problem need to be *non-conflicting*, i.e., at any point both the plant and the supervisor can fulfil their liveness properties eventually. Due to this additional property of solutions, our transformation in result (I) achieves reactive modules which do not falsify the assumption. The reverse transformation (result (II)), however, only holds if the computed reactive module returns solutions which are non-conflicting and hence a solution to the initial SCT synthesis problem. We identify three sufficient conditions for the latter to hold, namely (a) topologically closed , (b) topologically closed input/output, and (c) strongly non-anticipating input/output plant (resp. environment) behaviours. In the course of establishing result (II)(c), we identify a close connection between *strongly non-anticipating input/output plant behaviours* (see [Moor et al., 2011]) and *non-falsifiable environment assumptions* (see e.g. [Brenguier et al., 2017][2]) which ensure an almost identical form

[2] In [Brenguier et al., 2017], Sec. 3, this well known phenomenon in reactive synthesis is called *Win-under-Hype*.

of non-conflictingness of solutions and were derived independently in both communities.

Both synthesis algorithms are formalized as fixed-points in the μ-calculus. The RS algorithm uses a 3-nested fixed-point while SCT synthesis amounts to a 4-nested fixed-point iteration. Result (I) clarifies that this additional fixed-point iteration indeed generates an additional property on the solution. Notably, this additional property cannot be encoded as a ω-regular property and hence, the 4-nested fixed-point from SCT cannot result from a translation of an LTL synthesis problem into a μ-calculus formula. Regarding result (II), we may expect some computational benefits as a trade-off when imposing conditions (a)-(c) on the synthesis problem in SCT. Moreover, we show that for topologically closed plant (resp. environment) behaviours with alternating inputs and outputs both fixed-points collapse to the same 2-nested fixed-point. Using result (III)(b), the latter establishes equivalence of both algorithms in this case.

Related Work The question of how to synthesize solutions to an RS problem which do not falsify the assumptions has recently gained some attention from the reactive synthesis community, see e.g. [Chatterjee and Henzinger, 2007, Chatterjee et al., 2008, Chatterjee et al., 2010, Bloem et al., 2015, Brenguier et al., 2017]. Interestingly, it turns out that all these synthesis methods in general result in different solutions to a given reactive synthesis problem compared to our solution via result (I) based on the SCT perspective; see Section 5 for a detailed discussion.

Our study complements the recent comparison between RS and SCT by [Ehlers et al., 2017]. There, the authors focus attention on SCT over $*$-languages and discuss *maximal permissiveness* of a solution to the synthesis problem. This contrasts our choice of ω-languages, where a maximally permissive solution fails to exist in general and, taking a perspective common in RS, we resort to computing some solution provided that one exists. Moreover, [Ehlers et al., 2017] encode the requirement of a *non-conflicting* closed-loop configuration, as it is commonly discussed in the context of SCT, by a specific CTL formula and solve the synthesis problem by a specialised variant of RS. In contrast, to obtain our result (II), we address a non-conflicting closed loop by structural assumptions on the problem parameters which imply that for the corresponding RS problem the assumptions are *non-falsifiable* by any reactive module.

Although in general there exists no maximally permissive solutions to the synthesis problems discussed in this report, supervisory control theory for ω-languages provides a tight upper bound on all achievable closed-loop behaviours; see [Thistle and Wonham, 1994b]. In this regard our discussion relates to recent work in the reactive synthesis community which computes maximally permissive solutions using fair liveness requirements on edges to model liveness within these maximal solutions, see e.g. [Chatterjee et al., 2008, Klein et al., 2015].

Outline This report is structured as follows. After recalling necessary notation in Section 1, we give a concise introduction to reactive synthesis with environment assumptions and to supervisory control theory of ω-languages in Sections 2 and 3, respectively, including formal problem statements and a characterisation of their solutions in terms of ω-languages. The latter characterisation facilitates our formal comparison in Section 4, where our main technical results are obtained by (I) – Theorem 2 (p. 27), (II) – Theorem 3 (p. 30), Theorem 4 (p. 33), and Theorem 5

(p. 36), and (III) – Corollary 1 (p. 32), Corollary 2 (p. 33), and Corollary 3 (p. 36). The qualitative discussion of recent approaches to reactive synthesis that propose alternative means to handle environment assumptions in Section 5 concludes our study. Technical propositions and proofs are organised in the Appendices A, B and C.

1. Preliminaries

We introduce common terminology and recall some elementary facts regarding formal languages, automata and fixpoint calculus. A general introduction to these topics can be found in e.g. [Hopcroft and Ullman, 1979, Thomas, 1990, Bradfield and Stirling, 2006].

1.1. Formal Languages

Let Σ be a *finite alphabet*. The set of all $*$-*words* $s = \sigma_1\sigma_2\cdots\sigma_n$, $n \geq 1$, $\sigma_i \in \Sigma$ for all i, $1 \leq i \leq n$, is denoted Σ^+. The *empty string* is denoted ϵ, $\epsilon \notin \Sigma$, and we write $\Sigma^* := \Sigma^+ \dot\cup \{\epsilon\}$. An ω-*word* over Σ is an infinite sequence $\alpha = \sigma_1\sigma_2\sigma_3\cdots$ of symbols where we write $\alpha(i) := \sigma_i \in \Sigma$ to denote the i-th symbol, $i \in \mathbb{N}$. The set of all ω-words over Σ is denoted Σ^ω. We define $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. The subsets $L \subseteq \Sigma^*$ and $\mathcal{L} \subseteq \Sigma^\omega$ are called the $*$- *and* ω-*languages over* Σ, respectively.

For two words $s \in \Sigma^*$ and $t \in \Sigma^\infty$ we write $st \in \Sigma^\infty$ for the *concatenation*. As with all other operators in this paper, we take point-wise images for an extension to languages over Σ, e.g., we have $LM = \{st \,|\, s \in L,\ t \in M\}$ for $L,\ M \subseteq \Sigma^*$. For notational convenience, let $L^0 := \{\epsilon\}$, $L^{i+1} := (L^i)L$ and $L^* := \cup\{\, L^i \,|\, i \in \mathbb{N}\,\}$, where $L \subseteq \Sigma^*$, $L \neq \emptyset$. Likewise, let $L^\omega := \{\, s_1s_2s_3\cdots \in \Sigma^\omega \,|\, \forall\, i \in \mathbb{N}\colon s_i \in L\,\}$.

If, for two words $s \in \Sigma^*$ and $r \in \Sigma^\infty$ there exists $t \in \Sigma^\infty$ such that $st = r$, we say that s is a *prefix* of r, and we write $s \leq r$. If, in addition, $s \neq r$, we also write $s < r$. All prefixes of a word $t \in \Sigma^\infty$ are denoted $\operatorname{pfx} t \subseteq \Sigma^*$. For $L \subseteq \Sigma^*$, we have $L \subseteq \operatorname{pfx} L$, and, if equality holds, we say that L is *prefix closed*. Closedness of $*$-languages is retained under arbitrary union and under arbitrary intersection. Moreover, we say that L is *relatively closed w.r.t.* $M \subseteq \Sigma^*$, if $L = (\operatorname{pfx} L) \cap M$.

The *limit* $\lim L$ of $L \subseteq \Sigma^*$ contains all words $\alpha \in \Sigma^\omega$ which have infinitely many prefixes in L and we define $\operatorname{clo}\mathcal{L} := \lim \operatorname{pfx}\mathcal{L}$ as the *topological closure* of $\mathcal{L} \subseteq \Sigma^\omega$. An ω-language $\mathcal{L} \subseteq \Sigma^\omega$ is said to be *topologically closed* if $\mathcal{L} = \operatorname{clo}\mathcal{L}$ and *relatively topologically closed w.r.t.* $\mathcal{M} \subseteq \Sigma^\omega$, if $\mathcal{L} = (\operatorname{clo}\mathcal{L}) \cap \mathcal{M}$. This notion of closedness indeed defines a topology, i.e., $\emptyset$ and Σ^ω are closed, finite unions of closed ω-languages are closed, and arbitrary intersections of closed ω-languages are closed.

For $\Psi \subseteq \Sigma$, the *natural projection* p_Ψ of a word $s \in \Sigma^*$ is defined by removing all symbols not from Ψ, i.e., $\mathrm{p}_\Psi\,\epsilon = \epsilon$; and $\mathrm{p}_\Psi(s\sigma) = \mathrm{p}_\Psi(\sigma)$ for all $\sigma \in \Psi$, $s \in \Sigma^*$; and $\mathrm{p}_\Psi(s\sigma) = \mathrm{p}_\Psi\, s$ for all $\sigma \in \Sigma - \Psi$, $s \in \Sigma^*$. For the projection $\mathrm{p}_\Psi\,\alpha$ of an ω-word $\alpha \in \Sigma^\omega$, we consider the set $\mathrm{p}_\Psi \operatorname{pfx}\alpha$. Either there exists a unique maximal-length element $r \in \mathrm{p}_\Psi \operatorname{pfx}\alpha$, and we define $\mathrm{p}_\Psi\,\alpha := r \in \Psi^*$. Or, we have $\lim \mathrm{p}_\Psi \operatorname{pfx}\alpha = \{\beta\}$ for some $\beta \in \Psi^\omega$, and we let $\mathrm{p}_\Psi\,\alpha := \beta \in \Psi^\omega$. This concludes the definition of the operator p_Ψ on Σ^∞.

1.2. Automata

An *automaton* over the alphabet Σ is a tuple $M = (Q,\, \Sigma,\, \delta,\, Q_\mathrm{o})$ with *state set* Q, *transition relation* $\delta \subseteq Q \times \Sigma \times Q$ and set of *initial states* $Q_\mathrm{o} \subseteq Q$. M is called finite if Q is finite. We identify δ with its respective set-valued map $\delta : Q \times \Sigma \rightsquigarrow Q$ where $\delta(q, \sigma) := \{\, q' \,|\, (q, \sigma, q') \in \delta\,\}$, and with the common inductive extension to a word-valued second argument $s \in \Sigma^*$ by $\delta(q,\, \epsilon) := \{q\}$ and $\delta(q,\, s\sigma) := \delta(\,\delta(q, s),\, \sigma)$. Likewise, we denote the set of enabled events $\mathsf{Enab}(q) := \{\, \sigma \in \Sigma \,|\, \delta(q,\, \sigma) \neq \emptyset\,\}$ for $q \in Q$. If $|Q_\mathrm{o}| \leq 1$ and $|\delta(q, s)| \leq 1$ for all $q \in Q$, $s \in \Sigma^*$, then M is said to be *deterministic*. For deterministic automata, we interpret δ as partial function

and write $\delta(q, s) = q'$ and $\delta(q, s)!$ as short forms for $\delta(q, s) = \{q'\}$ and $\delta(q, s) \neq \emptyset$, respectively. A state $q \in Q$ is *reachable* if there exists a word $s \in \Sigma^*$ such that $q \in \delta(Q_\mathrm{o}, s)$. If all states are reachable, we say that the automaton M is reachable. Likewise, a state $q \in Q$ does not *deadlock* if there exists $\sigma \in \Sigma$ such that $\delta(q, \sigma) \neq \emptyset$.

We define $L = \mathrm{L}^*(M) := \{ s \in \Sigma^* \mid \delta(Q_\mathrm{o}, s) \neq \emptyset \}$ and $\mathcal{L} = \mathrm{L}^\omega(M) := \{ \alpha \in \Sigma^\omega \mid \operatorname{pfx} \alpha \subseteq \mathrm{L}^*(M) \}$ as the $*$- and ω-languages *generated* by M, respectively, which are prefix closed and topologically closed, respectively. Any finite automaton M can be transformed to a finite deterministic automaton that generates the same languages.

Generated languages can be restricted by *acceptance conditions*. For $*$-languages, we refer to a set of *final states* $F \subseteq Q$ and the extended automaton tuple $M = (Q, \Sigma, \delta, Q_\mathrm{o}, F)$, to define the *accepted* $*$-language of M by $\mathrm{L}_\mathrm{m}^*(M) := \{ s \in \Sigma^* \mid \delta(Q_\mathrm{o}, s) \cap F \neq \emptyset \}$. In this case, a state $q \in Q$ is called *coreachable* if there exists a word $s \in \Sigma^*$ such that $\delta(q, , s) \cap F \neq \emptyset$. The automaton M is coreachable if all states are coreachable. If M is reachable and coreachable, it is called *trim*. *Regular $*$-languages* are those that are accepted by a finite automaton, where requiring the automaton to be deterministic is not restrictive.

For ω-languages, we refer to an *acceptance condition* $\mathcal{F}$ and the extended automaton tuple $M = (Q, \Sigma, \delta, Q_\mathrm{o}, \mathcal{F})$. We begin with the common *Büchi acceptance condition*, where $\mathcal{F} \subseteq Q$ and where a *run* π over M to be accepted must visit $\mathcal{F}$ infinitely often. Technically, a run π over M is an infinite sequence of states $q_1 q_2 q_3 \cdots \in Q^\omega$ and it corresponds to the ω-word $\alpha = \sigma_1 \sigma_2 \sigma_3 \cdots \in \Sigma^\omega$, if $q_1 \in Q_\mathrm{o}$ and $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}$. The set of states that occur infinitely often in π is denoted $(\operatorname{Inf} \pi)$ and, hence, π is accepted if $(\operatorname{Inf} \pi) \cap \mathcal{F} \neq \emptyset$. The accepted ω-language $\mathrm{L}_\mathrm{m}^\omega(M)$ consists of all words $\alpha \in \Sigma^\omega$ for which there exists a corresponding accepted run over M. For deterministic Büchi automata, we have $\mathrm{L}_\mathrm{m}^\omega(M) = \lim \mathrm{L}_\mathrm{m}^*(M)$.

More general forms of acceptance conditions used in this paper occur with *generalized Büchi automata* and *parity automata*. For the former, we have that $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$ is a family of state sets $F_i \subseteq Q$, and a run π is accepted if $(\operatorname{Inf} \pi) \cap F_i \neq \emptyset$ for all $i \in \{1, \ldots, k\}$. For parity automata, the acceptance condition is given by a set of colours $\mathcal{F} = \{C_1, C_2, \ldots, C_k\}$ which is defined by a colouring function $c : Q \to \{1, ..., k\}$, s.t. $C_k = \{q \in Q \mid c(q) = k\}$, and a run π is accepted if the highest colour visited infinitely often is even, i.e., if $\max \operatorname{Inf} c(\pi)$ is even. As with Büchi automata, the set of ω-words corresponding to accepted runs is referred to as the accepted ω-language. Referring to an acceptance condition, we say that the automaton M is *trim* if for every state q there exists an accepted run that passes q at least once. The class of ω-languages that is accepted by some finite Büchi, generalized Büchi or parity automaton is referred to as the ω-regular languages. The class of ω-languages that is accepted by some *deterministic* finite Büchi automaton is a strict subset of the ω-regular languages while any ω-regular language is accepted by some *deterministic* parity automaton.

1.3. Two-Player Games

Let $\Sigma = \Sigma^0 \cup \Sigma^1$ be a disjoint composition of symbols and let $M = (Q^0 \cup Q^1, \Sigma^0 \cup \Sigma^1, q_0, \gamma^0 \cup \gamma^1, \mathcal{F})$ be an automaton s.t. $q_0' \in Q^0$, $\gamma^0 \subseteq Q^0 \times \Sigma^0 \times Q^1$, $\gamma^1 \subseteq Q^1 \times \Sigma^1 \times Q^0$ and $T^1 \subseteq Q$, where $Q = Q^0 \cup Q^1$. Then the tuple $H = (Q^0, Q^1, \Sigma^0, \Sigma^1, \gamma^0, \gamma^1)$ defines the turn-based two player game graph induced by M, where Q^l, Σ^l and γ^l with $l \in \{0, 1\}$ are interpreted as the player l state set, alphabet and transition set, respectively. Given a Büchi (reps. parity) acceptance condition $\mathcal{F}$, we call the tuple

$(H, \mathcal{F})$ a Büchi (resp. parity) game. Any run π of M is called a play over H and π is called a *winning* play for $(H, \mathcal{F})$ if $\pi \in \mathrm{L}^{\omega}_{\mathrm{m}}(M)$.

Given a game graph H, a strategy for player 1 is a function $f^1 : (Q^0 \cdot Q^1)^+ \to \Sigma^1$; it is *memoryless* if $f^1(\nu \cdot q^1) = f^1(q^1)$ for all $\nu \in (Q^0 \cdot Q^1)^* \cdot Q^0$ and all $q^1 \in Q^1$. A play π is *compliant* with f^1 if for all $k \in \mathbb{N}$ with $\pi(k) \in Q^1$ holds that $\pi(k+1) = \delta^1(\pi(k), f^1(\pi|_{[0,k]}))$. The strategy f^1 is a *winning strategy* for player 1 in the game $(H, \mathcal{F})$ if all plays compliant with f^1 are winning for $(H, \mathcal{F})$.

1.4. Fixpoint Calculus

Algorithmic solutions to synthesis problems are commonly stated as iterations of monotone operators defined on the state set of an automaton computed from the problem description. The purpose of the iteration is either to successively gain "good" states or to successively remove "bad" states. In both cases, the iteration stops when a fixpoint of the respective operator is attained. Since the subsets of the state set form a complete lattice and since the operators under consideration are monotone, the Tarski-Knaster-Theorem implies that the fixpoints also form a complete lattice, and, in particular, that a greatest fixpoint and a least fixpoint uniquely exist. A common formal framework for algorithms which can be expressed as fixpoints on transition systems is the *modal μ-calculus* [Kozen, 1983,Emerson and Jutla, 1991]. For the purpose of this paper, we pragmatically use notational conventions from the μ-calculus since this enables a compact representation of synthesis algorithms.

Let Q denote a finite set and consider a *monotone operator* f, i.e., $f(P') \subseteq f(P'') \subseteq Q$ for all $P' \subseteq P'' \subseteq Q$. From finiteness of Q it follows that $f(\cup\{P_i \,|\, i \in \mathbb{N}\}) = \cup\{\, f(P_i) \,|\, i \in \mathbb{N}\}$, where $P_i \subseteq P_{i+1} \subseteq Q$ for all $i \in \mathbb{N}$. Likewise $f(\cap\{P_i \,|\, i \in \mathbb{N}\}) = \cap\{\, f(P_i) \,|\, i \in \mathbb{N}\}$, where $P_{i+1} \subseteq P_i \subseteq Q$. The latter two properties are referred to as $\cup$-*continuity* and $\cap$-*continuity*, respectively.

As mentioned above, monotonicity of f implies the unique existence of the least fixpoint. By $\cup$-continuity, the least fixpoint equals $\cup\{f^i(\emptyset) \,|\, i \in \mathbb{N}\}$ and can be obtained by the iteration $P_1 := \emptyset$, $P_{i+1} := P_i \cup f(P_i)$. Note that for the finite base set Q under consideration, monotonicity implies that 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, with evaluations depending of the second parameter P''. 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''.

In typical applications, the multi-argument operator g is given as an expression using boolean set operators and simple monotone operators f that are related to the transition structure of an automaton. Care must be taken that the overall expression satisfies relevant monotonicity requirements. For the purpose of this paper, we refer to μ-calculus formulae that are known to be well formed and we therefore omit a deeper discussion of this issue.

2. Reactive Synthesis

This section gives a concise introduction to reactive synthesis with environment assumptions, derives a formal problem statement, and a characterisation of its solutions in terms of ω-languages. For illustration purposes, we recall an algorithmic solution of the synthesis problem for the specific case where all relevant ω-languages are provided as deterministic Büchi automata.

2.1. Reactive Modules

A *reactive module* is a device that reads the values of *input variables* in order to assign values to *output variables*, and that, over time, does so once in every *computation cycle*. A reactive module is commonly represented as a function r that maps the sequence of past input readings $s \in U^+$, to the current output assignment $y \in Y$, i.e,

$$r : U^+ \to Y \,. \tag{1}$$

Considering infinitely many computation cycles, the interaction of a reactive module with its environment generates an infinite sequence $\alpha \in (UY)^\omega$ of alternating input readings and output assignments. Therefore, the *behaviour* of a reactive module $r : U^+ \to Y$ is defined as the ω-language $\mathcal{L}$ of all sequences α that comply with r over all computation cycles :

$$\mathcal{L} := \{\, \alpha \in (UY)^\omega \mid \forall\, s \in (U \cup Y)^*,\, y \in Y : sy < \alpha \;\Rightarrow\; y = r(\mathrm{p}_U\, s) \,\} \,. \tag{2}$$

Note that, by construction, the above behaviour is topologically closed (see also Lemma 1). If, in addition, r is implemented as a finite automaton, then $\mathcal{L}$ is ω-regular.[3]

For our subsequent discussion we will eliminate the explicit reference to the reactive module $r : U^+ \to Y$ by utilizing a more direct characterization of those languages $\mathcal{L}$ that qualify for a representation by Eq. (2). Referring to J. C. Willems *behavioural systems theory* [Willems, 1991], we adapt the notion of *input-output systems* to the special case of topologically closed languages and to the notation used in the present paper.[4]

Definition 1. Given two ω-languages $\mathcal{L}, \mathcal{M} \subseteq (UY)^\omega$ or $\mathcal{L}, \mathcal{M} \subseteq (YU)^\omega$ of alternating inputs and outputs, with non-empty ranges U and Y, $U \cap Y = \emptyset$, respectively, we say that

(i) *U is a locally free input for $\mathcal{L}$* if
$\forall s \in \mathrm{pfx}\,\mathcal{L},\, u', u'' \in U :\;\; su' \in \mathrm{pfx}\,\mathcal{L} \;\Rightarrow su'' \in \mathrm{pfx}\,\mathcal{L}$;

(ii) *U is a relatively locally free input for $\mathcal{L}$* w.r.t. $\mathcal{M}$ if
$\forall s \in \mathrm{pfx}\,\mathcal{L} \cap \mathrm{pfx}\,\mathcal{M},\, u', u'' \in U :\;\; su' \in \mathrm{pfx}\,\mathcal{L} \;\Rightarrow su'' \in \mathrm{pfx}\,\mathcal{L}$;

[3] Typical means of implementation considered in the literature are finite Mealy automata with input alphabet U and output alphabet Y.

[4] The original literature [Willems, 1991] addresses the time axis $\mathbb{R}$ and $\mathbb{Z}$, so there is no exact technical match to the situation presented here. However, considering topologically closed languages the concepts are closely related: our notion of a *locally free input* corresponds to Willems' notion of a *free input that does not anticipate the output*, Definitions VIII.1 and VIII.4; our notion of the output to locally process the input corresponds to Willems' definition of *the output processes the input*, Definition VIII.3.

(iii) the *output locally processes the input* if
$\forall s \in \operatorname{pfx}\mathcal{L},\, y', y'' \in Y :\ sy' \in \operatorname{pfx}\mathcal{L},\ sy'' \in \operatorname{pfx}\mathcal{L}\ \Rightarrow\ y' = y''$. □

The above notion of inputs and outputs enables the following characterization of behaviours associated with some reactive module.

Lemma 1. Let U and Y, $U \cap Y = \emptyset$, denote the non-empty ranges of inputs and outputs, respectively. For a reactive module $r : U^+ \to Y$, the associated behaviour $\mathcal{L} \subseteq (UY)^\omega$ defined by Eq. (2) is non-empty and possesses the following properties:

(RM1) $\mathcal{L}$ is topologically closed,

(RM2) U is a locally free input for $\mathcal{L}$, and

(RM3) the output locally processes the input.

Vice versa, if a non-empty language $\mathcal{L} \subseteq (UY)^\omega$ satisfies conditions (RM1) – (RM3), then there exists a reactive module $r : U^+ \to Y$ with associated behaviour $\mathcal{L}$ s.t. $r(v)$ is the unique element of the singleton set

$$\{\, y \in Y \mid \exists\, s \in (UY)^* U\ .\ \mathrm{p}_U\, s = v \wedge sy \in \operatorname{pfx}\mathcal{L} \,\} \tag{3}$$

for $v \in U^+$. □

2.2. Problem Statement

The problem commonly referred to as *reactive synthesis* is about the systematic design of a reactive module, henceforth also referred to as the *system*, that provides a formal *guarantee* $\mathcal{G} \subseteq (UY)^\omega$. In the basic setting of reactive synthesis, it is assumed that any input symbol may be generated by the environment at any time and that, in turn, the environment accepts any output symbol generated by the system. Then, properties (RM2) and (RM3) of $\mathcal{L}$ ensure that the interaction of the system with its environment can be continued for infinitely many computation cycles, i.e., the two components *do not deadlock*. Thus, we end up with an ω-word $\alpha \in \mathcal{L}$. In turn, the system to provide the guarantee $\mathcal{G}$ amounts to the language inclusion specification $\mathcal{L} \subseteq \mathcal{G}$. The crucial point here is that $\mathcal{G}$ is assumed to be ω-regular, but, in contrast to $\mathcal{L}$, in general fails to be topologically closed.

In many applications, an arbitrary behaviour of the environment is considered unrealistic, and one explicitly accounts for formal assumptions imposed on the environment. For the purpose of our discussion, we parametrise such assumptions by an ω-language $\mathcal{A} \subseteq (UY)^\omega$ to express that (i) after each computation cycle the environment generates an input symbol $u \in U$ that complies with $\mathcal{A}$ in the sense that $su \in \operatorname{pfx}\mathcal{A}$, where $s \in (UY)^*$ denotes the word generated so far, and that (ii) over infinitely many computation cycles an ω-word $\alpha \in \mathcal{A}$ is generated. For *the system and the environment not to deadlock* we refer to assumption (i), (RM2) and (RM3), and formally require that

$$\forall s \in (\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L})\ .\ \exists \sigma \in U \cup Y\ .\ s\sigma \in (\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L}). \tag{4}$$

Regarding the guarantee $\mathcal{G}$ to be provided by the system, we refer to assumption (ii) and require that $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G} := ((U \cup Y)^\omega - \mathcal{A}) \cup \mathcal{G}$. This amounts to the following problem statement for the synthesis of a reactive module.

Problem 1 (Reactive Synthesis under Environment Assumptions)**.** Given two non-empty finite sets of *input symbols* U and *output symbols* Y, $U \cap Y = \emptyset$, an *environment assumption* $\mathcal{A} \subseteq (UY)^\omega$ and a *guarantee* $\mathcal{G} \subseteq (UY)^\omega$, the *reactive synthesis problem* $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$ asks to either construct a system such that the associated behaviour $\mathcal{L}$ does not deadlock with $\mathcal{A}$, see Eq. (4), and such that

$$\emptyset \neq \mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}\,, \tag{5}$$

or, to verify that no such system exists. □

Note that for $\mathcal{A} = \emptyset$ the upper bound $\mathcal{A} \to \mathcal{G}$ degenerates and the specification becomes $\mathcal{L} \subseteq (UY)^\omega$. Thus, in our discussion of the above problem we may whenever convenient assume that $\mathcal{A} \neq \emptyset$ and, likewise, $\mathcal{G} \neq (UY)^\omega$. Moreover, we have that $\mathcal{A} \to (\mathcal{G} \cap \mathcal{A}) = \mathcal{A} \to \mathcal{G}$, and, hence, we can restrict our discussion without loss of generality to the case where $\emptyset \neq \mathcal{G} \subseteq \mathcal{A} \subseteq (UY)^\omega$. Finally, we can choose $\mathcal{A} = (UY)^\omega$ to recover the basic setting without assumptions from our formal problem statement, i.e., in this case, Eq. (4) is trivially satisfied and the system to provide the guarantee collapses to the simple inclusion $\mathcal{L} \subseteq \mathcal{G}$.

With Lemma 1, the problem of reactive synthesis amounts to the construction of a non-empty subset $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$ that satisfies (RM1) – (RM3) and that does not deadlock, Eq. (4), or to the verification that no such subset exists. Henceforth, we may refer to a qualifying behaviour $\mathcal{L}$ as a *solution* of the synthesis problem. For practical reasons, one may additionally assume that the parameters $\mathcal{A}$ and $\mathcal{G}$ are ω-regular and, in turn, ask for an ω-regular solution $\mathcal{L}$ to derive a finite automaton realisation of the reactive module r. We conclude this section by commenting on how our technical problem statement relates to the literature.

Remark 1. The problem of reactive synthesis is more commonly formalized by using specifications given in linear temporal logic (LTL) over a set of atomic propositions $\mathcal{U} \cup \mathcal{Y}$ (see, e.g., [Pnueli and Rosner, 1989]). Such an LTL formula φ over $\mathcal{U} \cup \mathcal{Y}$ can be translated into a specification language $\mathcal{G} \subseteq (UY)^\omega$ with $U = 2^{\mathcal{U}}$ and $Y = 2^{\mathcal{Y}}$ by constructing a Büchi automaton from φ, and then, for algorithmic reasons, determinizing this automaton to obtain a deterministic Rabin or Parity automaton. The language accepted by the latter corresponds to the guarantee $\mathcal{G}$ in Problem 1. This transformation is well understood [Vardi and Wolper, 1986, Safra, 1988]; see e.g. [Finkbeiner, 2016] for a comprehensive discussion.

Remark 2. Environment assumptions are usually formalized in the reactive synthesis literature by an LTL formula which can be translated to a Büchi automaton (see Remark 1) or are directly given by an automaton model. The generated language $A_{\text{loc}} \subseteq (UY)^*$ of this automaton is always a superset of $\operatorname{pfx}\mathcal{A}$, but not necessarily identical to the latter. In the case of $\operatorname{pfx}\mathcal{A} \subsetneq A_{\text{loc}}$, the requirement of the system and the environment not to deadlock, Eq. (4), is substituted by

$$\forall s \in A_{\text{loc}} \cap (\operatorname{pfx}\mathcal{L})\ .\ \exists \sigma \in U \cup Y\ .\ s\sigma \in A_{\text{loc}} \cap (\operatorname{pfx}\mathcal{L}). \tag{6}$$

For the purpose of this report we observe that $\operatorname{pfx}\mathcal{A} \subsetneq A_{\text{loc}}$ only generates different solutions to Problem 1, whenever (5) is fulfilled by a string α which is not in $\mathcal{A}$, i.e., falsifies the assumption. As our comparison only targets solutions which do not have the latter property, assuming $\operatorname{pfx}\mathcal{A} = A_{\text{loc}}$ is not restrictive for our discussion.

2.3. Algorithmic Solution

The interaction of the system and its environment outlined above can be viewed as a turn-based two player game: in every round the environment player selects an arbitrary input $u \in U$ and the system selects the output $y \in Y$ according to r. It was shown by [Gurevich and Harrington, 1982, Pnueli and Rosner, 1989] that for ω-regular specifications there exists a winning strategy for the system player in this game if and only if the reactive synthesis problem has an ω-regular solution $\mathcal{L}$. Based on this result, a solution can be obtained by constructing a deterministic game, finding a winning strategy for the system player and translating this strategy into a finite automaton representing the reactive module. For a concise presentation of this construction, we consider the special case in which both $\mathcal{G}$ and $\mathcal{A}$ are realisable as deterministic Büchi automata. It should be noted that this does not imply that $\mathcal{A} \to \mathcal{G}$ can be realized by a deterministic Büchi automaton. However, it was shown in [Bloem et al., 2012] that there is nevertheless a direct and simple solution procedure for this case, which we briefly recall.

We refer to the previous section and restrict, without loss of generality, the discussion to the case of $\emptyset \neq \mathcal{G} \subseteq \mathcal{A}$. Given this setting, we consider a generalized Büchi automaton M with acceptance condition $\mathcal{F} = \{T^0, T^1\}$ s.t. $\mathcal{A} = \mathrm{L}^{\omega}_{\mathrm{m}}(M_{\mathcal{A}})$, $\mathcal{G} = \mathrm{L}^{\omega}_{\mathrm{m}}(M_{\mathcal{G}})$ and $\mathrm{pfx}(\mathcal{A}) = \mathrm{L}^{*}(M)$, where $M_{\mathcal{A}}$ and $M_{\mathcal{G}}$ refer to the simple Büchi automaton obtained from M by using the single winning state set T^0 and T^1, respectively. Additionally, we assume that M does not deadlock[5]. We refer to $\mathcal{G} \subseteq \mathcal{A} \subseteq (UY)^{\omega}$ to observe that the alternation of input readings and output assignments induces a disjoint union decomposition of the state set and the transition relation, i.e., M can be defined by the tuple

$$M = (Q^0 \cup Q^1, U \cup Y, q_0, \gamma^0 \cup \gamma^1, \{T^0, T^1\}) \tag{7}$$

s.t. $q_0 \in Q^0$, $\gamma^0 \subseteq Q^0 \times U \times Q^1$, $\gamma^1 \subseteq Q^1 \times Y \times Q^0$ and $T^0, T^1 \subseteq Q$ with $Q = Q^0 \cup Q^1$.

The generalized Büchi automaton M defines the turn-based deterministic game graph $H = (Q^0, Q^1, U, Y, \gamma^0, \gamma^1)$. In the context of the reactive synthesis problem, player 0 and player 1 are the *environment and system player*, respectively, and a system player winning strategy must ensure that all plays on H that visit T^0 infinitely often, must also visit T^1 infinitely often. This can be expressed by the four-colour parity game $(H, \mathcal{C})$ with $\mathcal{C} = \{\emptyset, Q \setminus T^0, T^0, T^1\}$, where $C_2 = Q \setminus T^0$ and $C_4 = T^1$ are the sets with even colour. Hence, a play π according to α on H is winning for $(H, \mathcal{C})$ if either T^0 is not visited infinitely often, i.e. $\alpha \notin \mathcal{A}$ or, else, if T^0 is visited infinitely often, then T^1 is also visited infinitely often, i.e., $\alpha \in \mathcal{A} \cap \mathcal{G}$. Both cases together amount to $\alpha \in \mathcal{A} \to \mathcal{G}$. Note also that, by construction, we have $\alpha \in \mathrm{clo}\,\mathcal{A}$ for any play α on H.

It was shown in [Emerson and Jutla, 1991] that the winning states for the system player in the 4-colour parity game can be computed by the fixed-point

$$\mathsf{Win}^1 = \nu X_4.\mu X_3.\nu X_2.\mu X_1.\bigcup_{k=1}^{4} \left(C_k \cap \mathsf{Pre}^1(X_k)\right),$$

[5] Given a trim deterministic Büchi automaton M^1 that accepts $\mathcal{G}$, we extend the state set by a not-accepting *dump-state* to obtain a full transition function. We then use the common product composition with a deterministic Büchi automaton M^0 that accepts $\mathcal{A}$ to obtain M, where the acceptance condition $\mathcal{F} = \{T^0, T^1\}$ is defined by the respective state component to be an accepted state in the automaton M^0 or M^1, respectively.

where $\mathsf{Pre}^1 : 2^Q \to 2^Q$ is the player 1 controllable prefix, defined for a set $A \subseteq Q$ by

$$\mathsf{Pre}^1(A) = \left\{q^0 \in Q^0 \middle| \forall u \in U \,.\, \delta^0(q^0, u) \in A\right\} \cup \left\{q^1 \in Q^1 \middle| \exists y \in Y \,.\, \delta^1(q^1, y) \in A\right\}. \tag{8}$$

However, as C_1 is empty, we can actually obtain the three-nested fixed point

$$\mathsf{Win}^1 = \nu X_4.\mu X_3.\nu X_2. \bigcup_{k=2}^{4} \left(C_k \cap \mathsf{Pre}^1(X_k)\right). \tag{9}$$

If $q_0 \in \mathsf{Win}^1$, the synthesis problem has a non-empty solution and a memoryless winning strategy for the system player can be derived from the iterations in (9) as follows. Consider the last iteration of the fixed-point in (9) resulting in the set $X_4^\infty = \mathsf{Win}^1 \subseteq Q$ and assume that we have to iterate over X_3 k-times before this fixed-point is reached. If X_3^i is the set obtained after the i-th iteration, we have that $X_4^\infty = \bigcup_i^k X_3^i$ with $X_3^i \subseteq X_3^{i+1}$, $X_3^0 = \emptyset$ and $X_3^k = X_3^\infty$. This defines a ranking for every state $q \in X_4^\infty$ s.t.

$$\mathsf{rank}(q) = i \quad \text{iff} \quad q \in X_3^i \setminus X_3^{i-1} \quad \text{for} \quad 1 \leq i \leq k. \tag{10}$$

Then $f^1 : Q^1 \cap X_4^\infty \to Y$ is a winning strategy for the system player in the Parity game $(H, \mathcal{C})$ if $y = f^1(q)$ implies $\mathsf{rank}(\delta^1(q, y)) < \mathsf{rank}(q)$ if $\mathsf{rank}(q) > 1$ and $\delta^1(q, y) \in X_4^\infty$ otherwise. It should be observed that f^1 defines r in (1) in the obvious way s.t. $r(\mathrm{p}_U\, s) = f^1(q)$ iff $\delta(q_0, s) = q$. A finite automaton realizing r (and therefore $\mathcal{L}$) is obtained by pruning M from all states $q \notin X_4^\infty$ and all transitions (q, y, q') s.t. $y \notin f^1(q)$. The proof that this winning strategy defines a reactive module solving Problem 1 can be obtained as a special case of the construction presented in [Bloem et al., 2012] and is therefore omitted.

Remark 3. Referring back to Remark 2, the outlined synthesis algorithm generalizes to the case where $\mathcal{G}$ is not necessarily a subset of $\mathcal{A}$ and $\mathrm{pfx}\,\mathcal{A} \subseteq A_{\mathrm{loc}}$. By following the same construction as in footnote 5, we obtain an automaton M which accepts $\mathcal{A} = \mathrm{L}_{\mathrm{m}}^\omega(M_{\mathcal{A}})$ and $\mathcal{G} \cap (\lim A_{\mathrm{loc}}) = \mathrm{L}_{\mathrm{m}}^\omega(M_{\mathcal{G}})$ and generates $A_{\mathrm{loc}} = \mathrm{L}^*(M)$. In [Bloem et al., 2012] M is directly obtained from a particular fragment of LTL s.t. these properties are satisfied and $Q = U \cup Y$. Our formalization is slightly more general in terms of the allowed state space. However, it is more restrictive in terms of acceptance conditions for $\mathcal{A}$ and $\mathcal{G}$; we only allow Büchi acceptance conditions while the algorithm in [Bloem et al., 2012] allows for *generalized* Büchi acceptance conditions. The resulting game over H is called a *General Reactivity Game of Rank 1* (GR(1) game for short) which can be solved by a vector version of (9).

Remark 4. In the special case where $\mathcal{A}$ is topologically closed, we can assume without loss of generality that $T^0 = Q$ and, hence, $C_2 = \emptyset$ and $C_3 = Q$. Then, the synthesis formula in (9) simplifies to

$$\mathsf{Win}^1(\mathcal{C}) = \nu X_4.\mu X_3.\mathsf{Pre}^1(X_3) \cup (T^1 \cap \mathsf{Pre}^1(X_4)). \tag{11}$$

This observation can be equally motivated by noting that for $T^0 = Q$, the Parity game $(H, \mathcal{C})$ reduces to $(H, \{Q \setminus T^1, T^1\})$ which is equivalent to the Büchi game (H, T^1). It is well known that Büchi games (H, T^1) are solvable by the fixed-point in (11); see e.g. [Maler et al., 1995, Zielonka, 1998]. In this context, the basic version of reactive synthesis without environment assumptions results in computing

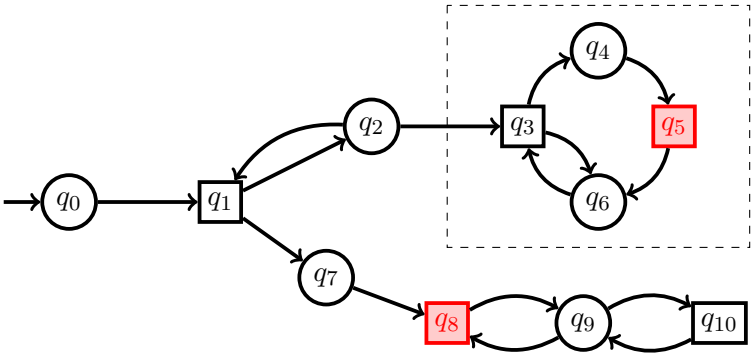

Figure 1: Transition structure of the Büchi automaton M in (7) discussed in Example 1. Environment and system states Q^0 and Q^1 are indicated by circles and squares, respectively. The final states $T^1 = \{q_5, q_8\}$ are indicated in red and $T^0 = Q^0$ is unrestricted, giving a topologically closed assumption $\mathcal{A}$. All winning states Win^1 for Problem 1 are contained in the dashed square.

a system winning strategy in the Büchi game (H', F^1) where H' is the game graph obtained from M^1 (given that $\mathcal{G}$ is realizable by the deterministic Büchi automaton M^1). In other words, adding a topologically closed environment assumption $\mathcal{A}$, algorithmically amounts to solving the basic reactive synthesis problem via Eq (11) over M in (7) instead of M^1.

2.4. Examples and Discussion

We illustrate the synthesis procedure of a reactive module via the algorithm presented in Section 2.3 by examples which outline some particularities of the algorithm and its solution.

Example 1. We first consider a topologically closed assumption $\mathcal{A}$ as discussed in Lemma 4 s.t. $\operatorname{pfx}\mathcal{A}$ is the language generated by M depicted in Figure 1 and $\mathcal{G}$ is accepted by M with final state set $T^1 = \{q_5, q_8\}$ (indicated in red in Figure 1). It was discussed in Remark 4 that this synthesis problem can be solved by the two nested fixed-point in (11). Considering the $j+1$-th iteration over X_4, we can compute X_4^{j+1} by evaluating the innermost fixed-point over X_3 using X_4^j. This is done by iteratively computing

$$X_3^0 = \emptyset \quad \text{and} \quad X_3^i = \mathsf{Pre}^1(X_3^{i-1}) \cup (F^1 \cap \mathsf{Pre}^1(X_4^j)) \cup X_3^{i-1}, \tag{12}$$

initialized with $X_4^0 = Q$. As $\mathsf{Pre}^1(Q) = Q$ for this example, we obtain the sequence

$$\begin{aligned}
X_3^1 &= F^1 = \{q_5, q_8\},\\
X_3^2 &= \mathsf{Pre}^1(X_3^1) \cup X_3^1 = \{q_4, q_7\} \cup X_3^1,\\
X_3^3 &= \mathsf{Pre}^1(X_3^2) \cup X_3^2 = \{q_3, q_1\} \cup X_3^2,\\
X_3^4 &= \mathsf{Pre}^1(X_3^3) \cup X_3^3 = \{q_6, q_2, q_0\} \cup X_3^3 = Q \setminus \{q_9, q_{10}\}.
\end{aligned}$$

It should be noted that X_3^2 does not contain q_9 as this is an environment state and not all transitions of q_9 reach $X_3^1 = F^1$, and hence $q_9 \notin \mathsf{Pre}^1(q_8)$. Furthermore,

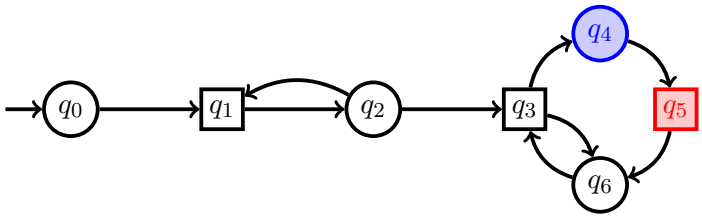

Figure 2: Transition structure of the Büchi automaton M in (7) discussed in Example 2. Environment and system states Q^0 and Q^1 are indicated by circles and squares, respectively. The final states $T^0 = \{q_4\}$ and $T^1 = \{q_5\}$ are indicated in blue and red, respectively.

observe that throughout the iteration over X_3, q_9 (and therefore also q_{10}) was never added to $X_{1,i}$ due to the same reason. The inner fixed point therefore terminates with $Q \setminus \{q_9, q_{10}\}$ and copies this set to X_4^1.

We now run a new round of computing (12) iteratively, initialized with X_4^1. As $\mathsf{Pre}^1(X_4^1) = Q \setminus \{q_8, q_9, q_{10}\}$ and hence $F^1 \cap \mathsf{Pre}^1(X_4^1) = \{q_5\}$, this results in the sequence

$$\begin{aligned} X_3^1 &= \{q_5\}, \\ X_3^2 &= \mathsf{Pre}^1(X_3^1) \cup \{q_5\} = \{q_4\} \cup X_3^1, \\ X_3^3 &= \mathsf{Pre}^1(X_3^2) \cup \{q_5\} = \{q_3\} \cup X_3^2, \\ X_3^4 &= \mathsf{Pre}^1(X_3^3) \cup \{q_5\} = \{q_6\} \cup X_3^3, \end{aligned}$$

i.e., returning the states contained in the dashed square depicted in Figure 1 (which is now copied to X_4^2). It should be noted that the iteration terminates for X_3^4 as the environment state q_2 has a successor (namely q_1) which is never added to X_3. Using the obtained set X_4^2, we see that $F^1 \cap \mathsf{Pre}^1(X_4^2) = \{q_5\}$ and the resulting inner fixed-point is the same as in the previous iteration. Hence the algorithm terminates with $\mathsf{Win}^1(F^1) = X_4^2$.

As $q_0 \notin \mathsf{Win}^1(F^1)$, Problem 1 has no solution in this example. However, if we would reduce M^1 to its sub-automaton contained in the dashed square depicted in Figure 1 and define q_3 as its initial state, the resulting system strategy would always choose the transition to q_4 in q_3 as $\mathsf{rank}(q_3) = 3$, $\mathsf{rank}(q_4) = 2$ and $\mathsf{rank}(q_6) = 4$. ◁

Example 2. We now consider a slightly different assumption $\mathcal{A}$ given by the language accepted by M depicted in Figure 2 with accepting state set $T^0 = \{q_4\}$ and the guarantee $\mathcal{G} \subseteq \mathcal{A}$ accepted by M with accepting state set $T^1 = \{q_5\}$. Similar to Example 1, we now evaluate the innermost fixed-point of $\mathsf{Win}^1(\mathcal{C})$ in (9) by iteratively computing

$$X_2^0 = Q \quad \text{and} \quad X_2^i = (T^1 \cap \mathsf{Pre}^1(X_4^k)) \cup (T^0 \cap \mathsf{Pre}^1(X_3^j)) \cup (Q \setminus T^0 \cap \mathsf{Pre}^1(X_2^{i-1})). \tag{13}$$

We initialize (13) with $X_4^0 = Q$ and $X_3^0 = \emptyset$. As $\mathsf{Pre}^1(Q) = Q$ and $\mathsf{Pre}^1(\emptyset) = \emptyset$ we obtain the sequence

$$\begin{aligned} X_2^1 &= T^1 \cup (Q \setminus T^0 \cap \mathsf{Pre}^1(Q)) = T^1 \cup (Q \setminus T^0 \cap Q) = Q \setminus T^0 \\ X_2^2 &= T^1 \cup (Q \setminus T^0 \cap \mathsf{Pre}^1(Q \setminus T^0)) = T^1 \cup (Q \setminus T^0 \cap Q) = Q \setminus T^0 = X_2^1. \end{aligned}$$

As $X_2^1 = X_2^2 = Q \setminus T^0$ the inner fixed-point terminates and we obtain $X_3^1 = Q \setminus T^0$. Using $\mathsf{Pre}^1(Q \setminus T^0) = Q$ (which was already used to compute X_2^2) we again evaluate (13) with $j = 1$ and $k = 0$ iteratively. This yields

$$X_2^1 = T^1 \cup T^0 \cup Q \setminus T^0 = Q = X_2^0$$

and hence $X_3^2 = Q = X_3^1$, implying $X_4^1 = Q = X_4^0$. Hence the fixed-point in (9) terminates with $\mathsf{Win}^1(\mathcal{C}) = Q$ on this example. Using the ranking function in (10) shows that all states in $Q \setminus T^0$ have rank 1 while q_4 has rank 2. This implies that the resulting winning strategy will trigger the transition from q_3 to q_6 instead of q_4. ◁

The problem discussed in Example 1 fails to have a solution, as the environment can prevent the system from reaching q_3 from where a system winning strategy exists. Interestingly, adding liveness to the environment using the restricted final state set $T^0 = \{q_4\}$ in Example 2 does not directly resolve this problem. The implication-style specification used in the formal statement of Problem 1 rather adds more sequences to the set of winning plays if $\mathcal{A}$ is not topologically closed; every sequence which does not conform with $\mathcal{G}$ (i.e., does not reach q_5 infinitely often) is still winning as long as it does not conform with the assumptions $\mathcal{A}$ (i.e., does not visit q_4 infinitely often) either. As this is true for the sequence iterating between q_1 and q_2 and preventing the system to reach q_3, the synthesis problem now has a solution.

This has another consequence. Looking at the states in the dashed box of Figure 1, we see that now choosing to transition to q_4 or q_6 from q_3 is equally good. While always choosing the former ensures to win by visiting both q_4 and q_5 infinitely often, the latter ensures to win by visiting neither q_4 nor q_5 at all. As there is no way to distinguish these two choices from the iteration of the fixed-point, one cannot rank one over the other. Therefore, the ranking function in (10) simply assigns ranks along a shortest path which happens to favour a transition from q_3 to q_6 in this example. This results in a falsification of the assumption by the synthesized reactive module.[6]

[6] We will comment on recent advances in reactive synthesis targeting this problem in Section 5.

3. Supervisory Control

The purpose of this section is to give a concise introduction to supervisory control for non-terminating processes and to do so from a perspective and in a notation most convenient for a comparison to reactive synthesis. Technically, we refer to a branch of supervisory control theory proposed by [Ramadge, 1989, Thistle and Wonham, 1994b] which explicitly accounts for non-terminating processes and therefore utilises ω-languages as the base model. For illustration purposes, we again give an algorithmic solution of supervisory controller synthesis for the specific case of deterministic Büchi automata realizations of the involved ω-languages.

3.1. Supervisors

A *supervisory controller* is a device that takes as input a finite sequence of events from an alphabet Σ generated by a process which is commonly referred to as the *plant* and, in turn, outputs a *control pattern* $\gamma \subseteq \Sigma$. Formally, the supervisor is defined as a map

$$f : \Sigma^* \to \Gamma\,, \quad \text{with} \quad \Gamma := \{\, \gamma \subseteq \Sigma \,|\, \Sigma_{\text{uc}} \subseteq \gamma \,\}\,, \tag{14}$$

where $\Sigma_{\text{uc}} \subseteq \Sigma$ are so called *uncontrollable events*. On start-up, the supervisor applies the control pattern $\gamma = f(\epsilon)$ and thereby restricts the plant to generate an event $\sigma \in \gamma$. After the plant has generated its event, the control pattern is updated accordingly, and so forth. In this process, the role of the uncontrollable events Σ_{uc} is that, by the definition of Γ, their occurrence cannot be prevented by the supervisor. From a reactive synthesis point of view, the supervisor is the system we seek to design and, for practical purposes, a realisation as a finite automaton is of a particular interest.

For the subsequent discussion, however, the following representation as an ω-language turns out more convenient:[7]

$$\mathcal{L} := \{\alpha \in \Sigma^\omega \,|\, \forall\, s \in \Sigma^*,\ \sigma \in \Sigma :\ s\sigma < \alpha \ \Rightarrow\ \sigma \in f(s)\,\}\,. \tag{15}$$

The language defined by Eq. (15) is referred to as the *behaviour associated with the supervisor* f. The following lemma characterises languages that match the behaviour of some supervisor.

Lemma 2. Given Σ, denote $\Sigma_{\text{uc}} \subseteq \Sigma$ the non-empty set of uncontrollable events. A behaviour $\mathcal{L} \subseteq \Sigma^\omega$ associated with some supervisor $f : \Sigma^* \to \Gamma$ is non-empty and exhibits the following properties:

(SC1) $\mathcal{L}$ is topologically closed, and

(SC2) $\mathcal{L}$ is *universally controllable*, i.e., $(\text{pfx}\,\mathcal{L})\Sigma_{\text{uc}} \subseteq \text{pfx}\,\mathcal{L}$.

Vice versa, if a non-empty language $\mathcal{L} \subseteq \Sigma^\omega$ satisfies (SC1) and (SC2), then $f : \Sigma^* \to \Gamma$ defined by

$$f'(s) := \{\, \sigma \in \Sigma \,|\, s\sigma \in \text{pfx}\,\mathcal{L}\,\} \cup \Sigma_{\text{uc}} \tag{16}$$

for $s \in \Sigma^*$ is a supervisor with associated behaviour $\mathcal{L}$. □

[7] The proposed representation of a supervisor f by the ω-language $\mathcal{L}$ does not account for supervisors which deadlock by themselves, i.e., supervisors that output an empty control pattern. However, assuming a non-empty Σ_{uc} is not restrictive and technically rules out the degenerated case of empty control patterns.

3.2. Problem Statement

The problem commonly referred to as *supervisory controller synthesis* is about the systematic design of a supervisor, such that the resulting closed-loop system – established by the feedback composition of this supervisor with the plant – satisfies a given specification. Referring to the reactive synthesis perspective, this identifies the plant as the environment of the system that we seek to design.

When the plant behaviour is given as an ω-language $\mathcal{A} \subseteq \Sigma^\omega$, the closed-loop configuration with a supervisor with associated behaviour $\mathcal{L} \subseteq \Sigma^\omega$ evolves on words that comply with both component behaviours. Technically, we distinguish the *local closed-loop behaviour*

$$K_{\mathrm{loc}} := (\mathrm{pfx}\,\mathcal{L}) \cap (\mathrm{pfx}\,\mathcal{A}) \tag{17}$$

and the *accepted closed-loop behaviour*

$$\mathcal{K} := \mathcal{L} \cap \mathcal{A}\,. \tag{18}$$

Regarding liveness of the closed-loop configuration, supervisory control commonly addresses not only deadlocks but also livelocks. The latter are characterised by finite sequences $s \in K_{\mathrm{loc}}$ from the local closed-loop behaviour that can be continued indefinitely within K_{loc} but any such infinite extension fails to satisfy the plant acceptance condition. Technically, we ask for a *non-blocking supervisor*, i.e., we require that $\mathcal{L}$ and $\mathcal{A}$ are *non-conflicting*:

$$(\mathrm{pfx}\,\mathcal{L}) \cap (\mathrm{pfx}\,\mathcal{A}) = \mathrm{pfx}(\mathcal{L} \cap \mathcal{A})\,. \tag{19}$$

In particular, the local behaviour K_{loc} of a non-conflicting closed loop can be recovered by $K_{\mathrm{loc}} = \mathrm{pfx}\,\mathcal{K}$ and we then refer to $\mathcal{K}$ concisely as the *closed-loop behaviour*.

In the absence of an acceptance condition of the plant, i.e., when $\mathcal{A}$ is topologically closed, livelocks are not an issue and, hence, non-conflictingness, Eq. (19), is equivalent to the *absence of deadlocks*, Eq. (4). Then, the generation of the infinite event sequence can be thought of purely in a step-by-step fashion.

The interpretation of the case when $\mathcal{A}$ is not topologically closed is more involved. Here, Eq. (19) guarantees that at any instance of time, the plant under supervision can achieve its acceptance condition on at least one infinite extension of the string generated so far. Thus, at some stage the plant must actually choose a path that not only attains a marked state but also complies with the restrictions subsequently imposed by the supervisor. In general, this should be interpreted as a form of cooperation. We will come back to this point in Section 4.3.

For the purpose of our discussion, we observe that an upper-bound specification $\mathcal{K} \subseteq \mathcal{G}$ can be interpreted as a guarantee in the reactive synthesis context, with the particular feature, that any supervisor enforcing this guarantee on the closed loop behaviour cannot invalidate the assumption $\mathcal{A}$ representing the plant behaviour. The latter observation is due to the definition of $\mathcal{K}$ in (18) implying $\mathcal{K} \subseteq \mathcal{A}$. Hence, if we establish a supervisor such that the closed-loop behaviour $\mathcal{K}$ is non-empty and satisfies $\mathcal{K} \subseteq \mathcal{G}$, we trivially obtain $\mathcal{K} \subseteq \mathcal{A} \cap \mathcal{G} \subseteq \mathcal{A} \to \mathcal{G}$.

We summarize the above discussion in the following formal statement of the synthesis problem for the supervision of non-terminating processes.

Problem 2 (Supervisory Controller Synthesis)**.** Given an alphabet Σ with the non-empty set of *uncontrollable events* $\Sigma_{\text{uc}} \subseteq \Sigma$, a *plant* $\mathcal{A} \subseteq \Sigma^\omega$ and an *upper-bound specification* $\mathcal{G} \subseteq \Sigma^\omega$, the *supervisory control problem* $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ asks to either construct a non-blocking supervisor with associated behaviour $\mathcal{L} \subseteq \Sigma^\omega$, see Eq. (19), such that

$$\emptyset \neq \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{G}\,, \tag{20}$$

or, to verify that no such supervisor exists. □

Referring to the behavioural characterisation of supervisors, Lemma 2, we identify a qualifying associated behaviour $\mathcal{L}$ as a solution to the control problem. Note that for $\mathcal{A} = \emptyset$ the problem trivially has no solution and that for $\Sigma_{\text{uc}} = \Sigma$ we have $\mathcal{L} = \Sigma^\omega$ for the only qualifying supervisor; i.e., in this case the synthesis problem collapses to the verification of $\mathcal{A} \subseteq \mathcal{G}$. Thus, in our discussion of supervisory controller synthesis problems, we may whenever convenient assume non-trivial problem parameters $\mathcal{A} \neq \emptyset$ and $\Sigma_{\text{uc}} \neq \Sigma$. As in the setting of reactive synthesis, Section 2.2, we can, without loss of generality, restrict our discussion to the case of $\emptyset \neq \mathcal{G} \subseteq \mathcal{A}$.

Remark 5. As with reactive synthesis, we may alternatively represent the plant behaviour by two distinct languages $A_{\text{loc}} \subseteq \Sigma^*$ and $\mathcal{A} \subseteq \Sigma^\omega$, where A_{loc} is prefix-closed and represents the local behaviour. In this regard, the literature [Ramadge, 1989, Thistle and Wonham, 1994b] specifically addresses the case $\operatorname{pfx}\mathcal{A} \subsetneq A_{\text{loc}}$ of a *blocking plant* and the supervisors task is to avoid livelocks and deadlocks in the closed loop. Since non-conflictingness is addressed by our formal problem statement, we can introduce a distinguished uncontrollable event $\dagger \notin \Sigma$ to make plant conflicts explicit; i.e., we substitute A_{loc} by $A_{\text{loc}} \cup ((A_{\text{loc}} - \operatorname{pfx}\mathcal{A})\dagger^*)$ and $\mathcal{A}$ by $\mathcal{A} \cup ((A_{\text{loc}} - \operatorname{pfx}\mathcal{A})\dagger^\omega)$, to formally obtain $A_{\text{loc}} = \operatorname{pfx}\mathcal{A}$. Using the original guarantee $\mathcal{G}$, the supervisor then must circumvent conflicts by implicitly avoiding the generation of the uncontrollable event $\dagger \notin \Sigma$. Using this pre-processing stage, our formal problem statement with $A_{\text{loc}} = \operatorname{pfx}\mathcal{A}$ is not restrictive.

3.3. Achievable Closed-Loop Behaviours

Given a plant, the common approach to solve Problem 2 is via a characterisation of all closed-loop behaviours that can be achieved by non-blocking supervisory control.[8] To this end, we refer to the following proposition from [Ramadge, 1989].

Proposition 1. Given an alphabet Σ with uncontrollable events $\Sigma_{\text{uc}} \subseteq \Sigma$, consider two languages $\mathcal{A}$ and $\mathcal{K}$ with $\emptyset \neq \mathcal{K} \subseteq \mathcal{A} \subseteq \Sigma^\omega$. Then there exists a non-blocking supervisor $f : \Sigma^* \to \Gamma$ for the plant $\mathcal{A}$ with closed-loop behaviour $\mathcal{K}$ if and only if

(i) $\mathcal{K}$ is relatively topologically closed w.r.t. $\mathcal{A}$, i.e., $\mathcal{K} = \mathit{clo}(\mathcal{K}) \cap \mathcal{A}$, and

(ii) $\mathcal{K}$ is $*$-*controllable* w.r.t. $\mathcal{A}$, i.e., $((\operatorname{pfx}\mathcal{K})\Sigma_{\text{uc}}) \cap (\operatorname{pfx}\mathcal{A}) \subseteq (\operatorname{pfx}\mathcal{K})$.

Given a closed-loop behaviour $\mathcal{K}$ that satisfies conditions (i) and (ii), a corresponding supervisor f can be extracted by

$$f(s) := \{\, \sigma \in \Sigma \mid s\sigma \in \operatorname{pfx}\mathcal{K} \,\} \cup \Sigma_{\text{uc}} \quad \text{for all } s \in \Sigma^*\,. \tag{21}$$

[8] The style of argument here is similar to the *Youla-Kučera parameterization* of all stabilising controllers for a linear time invariant system, which is conceived a major milestone in control theory; see e.g., [Kučera, 2011] for a recent presentation. For the situation here, a convenient characterisation of supervisors with the qualitative property not to block at the first stage, allows us to care about the quantitative upper bound specification at an largely independent second stage.

□

So far, the solution of the controller synthesis problem is effectively reduced to the synthesis of a non-empty closed-loop behaviour $\mathcal{K} \subseteq \mathcal{A} \cap \mathcal{G}$ satisfying conditions (i) and (ii) from Proposition 1. The existence of such a closed-loop behaviour $\mathcal{K}$ can be formally assessed by the following union construct

$$\mathcal{K}^{\uparrow} := \cup\{\, \mathcal{K} \subseteq \mathcal{A} \cap \mathcal{G} \,|\, \mathcal{K} \text{ satisfies (i) and (ii) from Proposition 1} \,\}\,. \tag{22}$$

Clearly, a solution exists if and only if $\mathcal{K}^{\uparrow}$ is non-empty. Moreover, if the above union itself exhibits conditions (i) and (ii) from Proposition 1 then $\mathcal{K}^{\uparrow}$ is referred to as the *supremal closed-loop behaviour* and a so called *maximally permissive supervisor* to solve the synthesis problem can be extracted from $\mathcal{K}^{\uparrow}$ via Eq. (21). However, the union in Eq. (22) in general fails to preserve topological closedness and hence, a maximally permissive solution does not exist in general. This contrasts SCT for $*$-languages but conforms with the situation for reactive synthesis.

Remark 6. Condition (i) in Proposition 1 is specific for the supervision of ω-languages, whereas condition (ii) is literally identical with the original notion of controllability introduced by [Ramadge and Wonham, 1987] for the more common setting of supervisory control where both $\mathcal{A}$ and $\mathcal{G}$ are $*$-languages. In that setting, all relevant closed-loop properties are preserved under arbitrary union and a supremal closed-loop behaviour uniquely exists. For ω-languages, it is only under the additional assumption that $\mathcal{A} \cap \mathcal{G}$ itself is relatively topologically closed w.r.t. $\mathcal{A}$, that $\mathcal{K}^{\uparrow}$ qualifies for an achievable closed-loop behaviour; see [Ramadge, 1989]. In this case, $\mathcal{K}^{\uparrow}$ can be computed by the common synthesis algorithm for $*$-languages from [Ramadge and Wonham, 1987] with appropriately chosen parameters $A \subseteq \Sigma^*$ and $G \subseteq \Sigma^*$ and with a minor variation to address deadlocks; see also the appendix of [Moor et al., 2012]. The relation between supervisory control of $*$-languages and reactive synthesis is discussed in detail by [Ehlers et al., 2017], with a particular focus on the supremal closed-loop behaviour and a corresponding maximally permissive supervisor.

To this end, [Thistle and Wonham, 1994b] introduce the following notion of the controllability prefix which, as we shall see, can be utilized to solve the controller synthesis problem.

Definition 2. Given an alphabet Σ with uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$ and a plant $\mathcal{A} \subseteq \Sigma^{\omega}$, consider the upper bound specification $\mathcal{G} \subseteq \Sigma^{\omega}$. The *controllability prefix of* $\mathcal{G}$ *w.r.t.* $\mathcal{A}$ is denoted $\mathrm{cfx}_{\mathcal{A}}\,\mathcal{G}$ and defined as the set of strings $s \in \mathrm{pfx}\,\mathcal{G}$, for which there exists $\mathcal{V} \subseteq \mathcal{A} \cap \mathcal{G} \cap (s\Sigma^{\omega})$ such that

(i) $\mathcal{V}$ is relatively topologically closed w.r.t. $\mathcal{A} \cap (s\Sigma^{\omega})$, i.e., $\mathcal{V} = \mathit{clo}(\mathcal{V}) \cap (\mathcal{A} \cap (s\Sigma^{\omega}))$, and

(ii) $\mathcal{V}$ is $*$-*controllable* w.r.t. $\mathcal{A} \cap (s\Sigma^{\omega})$, i.e., $((\mathrm{pfx}\,\mathcal{V})\Sigma_{\mathrm{uc}}) \cap (\mathrm{pfx}\,\mathcal{A}) \cap (s\Sigma^*) \subseteq (\mathrm{pfx}\,\mathcal{V})$.

When the role of the uncontrollable events is not clear from the context, we also write $\mathrm{cfx}_{\mathcal{A},\,\Sigma_{\mathrm{uc}}}\,\mathcal{G}$

Comparing the above conditions (i) and (ii) with Proposition 1, we see that $\mathcal{V}$ is a closed-loop behaviour that can be enforced by a non-blocking supervisor, if

it "takes over to control the plant" after the string s in the controllability prefix $\mathrm{cfx}_{\mathcal{A}}\,\mathcal{G}$ was generated by the plant. As $\mathcal{V} \subseteq \mathcal{G}$ this supervisor is able to enforce the guarantee $\mathcal{G}$. In [Thistle and Wonham, 1994b], the set $\mathrm{cfx}_{\mathcal{A}}\,\mathcal{G}$ is referred to as the "winning configurations of the supervisor" for a game theoretic interpretation. The cited literature further develops a notion of ω-controllability in terms of the controllability prefix, which leads to an alternative characterisation of $\mathcal{K}^{\uparrow}$. Clearly, Problem 2 has a solution if and only if $\epsilon \in \mathrm{cfx}_{\mathcal{A}}\,\mathcal{G}$. For ω-regular parameters, the controllability prefix can be represented in terms of a fixed-point similar to those used in the context of reactive synthesis. The corresponding iteration allows to extract a supervisor that solves the control problem, provided that a solution exists, and we recall the fixed-point characterisation of $\mathrm{cfx}_{\mathcal{A}}(\mathcal{G})$ in the following section.

3.4. Algorithmic Solution

We state a solution procedure under the assumption that $\emptyset \neq \mathcal{G} \subseteq \mathcal{A}$. By (20), this is not restrictive: if $\mathcal{G} = \emptyset$ the problem has no solution; and substitution of $\mathcal{G}$ by $\mathcal{A} \cap \mathcal{G}$ does not affect solutions. For the sake of a concise exposition, we also assume that we are given a trim generalized deterministic Büchi automaton[9]

$$M = (Q, \Sigma, q_0, \delta, \{F_{\mathcal{A}}, F_{\mathcal{G}}\}) \tag{23}$$

s.t. $\mathcal{A} = \mathrm{L}^{\omega}_{\mathrm{m}}(M_{\mathcal{A}})$, $\mathcal{G} = \mathrm{L}^{\omega}_{\mathrm{m}}(M_{\mathcal{G}})$ and $\mathrm{pfx}(\mathcal{A}) = \mathrm{L}^{*}(M)$ where $M_{\mathcal{A}}$ and $M_{\mathcal{G}}$ refer to the simple (and deterministic) Büchi automata obtained from M by using the single winning state set $F_{\mathcal{A}}$ and $F_{\mathcal{G}}$, respectively. We call M a representation of $\mathcal{A}$ and $\mathcal{G}$.

Given the generalized deterministic Büchi automaton M, a winning configuration $s \in \mathrm{cfx}_{\mathcal{A}}\,\mathcal{G}$ corresponds to the state $q = \delta(q_0, s)$ reachable by s from q_0 in M, and hence q is called a winning state. To compute the set of winning states, one needs to test for each state $q \in Q$ whether or not M can be constraint by control patterns such that any infinite run that starts in q and that visits $F_{\mathcal{A}}$ infinitely often is guaranteed to also visit $F_{\mathcal{G}}$ infinitely often. This test is organised as the four-nested fix-point

$$\mathsf{Win}(M) := \nu Z \,.\, \mu Y \,.\, \nu X \,.\, \mu W \,.\, \mathsf{Pre}((W \setminus F_{\mathcal{A}}) \cup Y \cup (F_{\mathcal{G}} \cap Z), X \setminus F_{\mathcal{A}}), \tag{24}$$

computing the set of winning states, where

$$\mathsf{Pre}(T,\, D) := \left\{ q \in Q \,\middle|\, \begin{pmatrix} \exists \sigma \in \Sigma \,.\, \delta(q,\sigma) \in T \\ \wedge \forall \sigma \in \Sigma_{\mathrm{uc}} \,.\, \delta(q,\sigma)! \;\Rightarrow\; \delta(q,\sigma) \in T \cup D) \end{pmatrix} \right\} \tag{25}$$

is the *inverse dynamics operator* Pre that maps a set of *target states* $T \subseteq Q$ w.r.t. a *domain constraint* $D \subseteq Q$ to its one-step-predecessors.

The fixed-point in (24) is derived from an algorithm with more general scope provided by [Thistle and Wonham, 1994a], which we adapted to address the special case in which both the plant $\mathcal{A}$ and the specification $\mathcal{G}$ are represented by deterministic Büchi automata. By the following theorem, Eq. (24) indeed establishes a representation of $\mathrm{cfx}_{\mathcal{A}}\,\mathcal{G}$ and we provide an independent and self-contained proof in Appendix B.

[9] If $\mathcal{A}$ and $\mathcal{G}$ are represented by Büchi automata $M_{\mathcal{A}}$ and $M_{\mathcal{G}}$, respectively, M can be computed as a result of a product composition of the two, followed by a removal of all states which are not reachable or not coreachable.

Theorem 1. Let Σ be an alphabet with the non-empty set of uncontrollable events $\Sigma_{\text{uc}} \subseteq \Sigma$, and let $\mathcal{A} \subseteq \Sigma^\omega$ be a plant behaviour and $\mathcal{G} \subseteq \mathcal{A}$ a non-empty upper-bound specification. If a deterministic generalized Büchi automaton $M = (Q, \Sigma, q_0, \delta, \{F_\mathcal{A}, F_\mathcal{G}\})$ is a representation of $\mathcal{A}$ and $\mathcal{G}$ then, for all $s \in \Sigma^*$,

$$s \in \text{cfx}_\mathcal{A}\mathcal{G} \Leftrightarrow \delta(q_0, s) \in \mathsf{Win}(M). \tag{26}$$

□

Given the sets of winning states $\mathsf{Win}(M)$ computed in (24) we are left with constructing a non-blocking supervisor. Recall that a solution exists if and only if $\epsilon \in \text{cfx}_\mathcal{A}\,\mathcal{G}$, which amounts to $q_0 \in \mathsf{Win}(M)$. We therefore assume that $q_0 \in \mathsf{Win}(M)$ and derive a candidate supervisor from iterations of the fixed-point in (24) as follows.

Consider the last iteration of the fixed-point in (24) resulting in the set $Z^\infty = \mathsf{Win}(M) \subseteq Q$ of states and assume that the fixed point over Y is reached after k iterations. If Y^i is the set obtained after the i-th iteration, we have that $Z^\infty = \bigcup_i^k Y^i$ with $Y^i \subseteq Y^{i+1}$, $Y^0 = \emptyset$ and $Y^k = Z^\infty$. Furthermore, let $X^i = Y^i$ denote the fixed-point of the iteration over X resulting in Y^i and denote by W_j^i the set obtained in the jth iteration over W performed while computing X^i. Then we have for all $0 \leq i \leq k$ that $Y^i = X^i = \bigcup_j^l W_j^i$ with $W_j^i \subseteq W_{j+1}^i$, $W_0^i = \emptyset$ and $W_l^i = Y^i$.

Using these sets, we define a ranking for every state $q \in Z^\infty$ s.t.

$$\mathsf{rank}(q) = \begin{cases} (i,j), & q \in \left(Y^i \setminus Y^{i-1}\right) \cap \left(W_j^i \setminus W_{j-1}^i\right),\ i,j > 0 \\ (0,0), & q \in Z^\infty \cap F_\mathcal{G} \end{cases} \tag{27}$$

initialized with $Y^0 := Z^\infty \cap F_\mathcal{G}$ and $W_0^i = \emptyset$ for all $0 < i \leq k$. We order ranks lexicraphically. Based on this ranking function we define a state feedback map $g : Z^\infty \to \Gamma$ s.t. for all $q \in Z^\infty$

$$g(q) := \begin{cases} \Sigma_{uc} \cup \{\sigma \in \Sigma_c \mid \mathsf{rank}(\delta(q,\sigma)) < \mathsf{rank}(q)\} & \text{if } \mathsf{rank}(q) > (0,0) \\ \Sigma_{uc} \cup \{\sigma \in \Sigma_c \mid \delta(q,\sigma) \in Z^\infty\} & \text{otherwise.} \end{cases} \tag{28}$$

The state feedback map g defines a supervisor $f : \Sigma^* \to \Gamma$ in the obvious way, i.e. $f(s) = g(q)$ if $\delta(q_0, s) = q$ and $f(s) = \emptyset$, otherwise. The behaviour $\mathcal{L}$ associated with f is defined via (15). Given this supervisor we have the following soundness result which is proven by Lemma 5 and Lemma 6 in Section B.

Proposition 2. Let Σ be an alphabet with the non-empty set of uncontrollable events $\Sigma_{\text{uc}} \subseteq \Sigma$, and let $\mathcal{A} \subseteq \Sigma^\omega$ be a plant behaviour and $\mathcal{G} \subseteq \mathcal{A}$ an upper-bound specification, s.t. M in (23) is a representation of $\mathcal{A}$ and $\mathcal{G}$. If $q_0 \in \mathsf{Win}(M)$, the supervisor f constructed via g in (28) with associated behaviour $\mathcal{L}$ solves Problem 2, i.e., f is a non-blocking supervisor with $\emptyset \neq \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{G}$. □

Remark 7. The state feedback g, as defined in Eq. (28), will only enable controllable events for transitions that decrease the rank and, hence, achieve progress in terms of attaining a marked state. Regarding the acceptance condition, however, it is sufficient to attain markings eventually. A more permissive feedback is obtained by initially controlling the local closed-loop K_{loc} to be a subset of $\text{cfx}_\mathcal{A}\,\mathcal{G}$ and only eventually to activate the supervisor constructed above. The original literature [Thistle and Wonham, 1994b] addresses permissiveness by explicitly considering a lower-bound specification $\mathcal{E}$, $\emptyset \neq \mathcal{E} \subseteq \mathcal{G}$. Under the condition that $\mathcal{E}$ is relatively

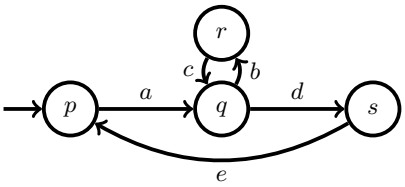

Figure 3: Transition structure of automaton M in (23) representing $\mathcal{A}$ and $\mathcal{G}$ for the instance of Problem 2 which is solved in Example 3.

topologically-closed w.r.t. $\mathcal{A}$ and that $\mathcal{E} \subseteq \mathcal{K}^{\uparrow}$, a supervisor can be constructed such that the closed-loop behaviour $\mathcal{K}$ satisfies $\mathcal{E} \subseteq \mathcal{K} \subseteq \mathcal{G}$. Thus, the additional problem parameter $\mathcal{E}$ can be used to tune permissiveness of the supervisor. As mentioned in Remark 6, if $\mathcal{G} \cap \mathcal{A}$ is relatively topologically-closed w.r.t. $\mathcal{A}$ then so is $\mathcal{K}^{\uparrow}$. In this case, one can choose $\mathcal{E} = \mathcal{K}^{\uparrow}$ and the supervisor constructed by [Thistle and Wonham, 1994b] essentially matches the one that can be obtained by synthesis procedures from $*$-languages.

Remark 8. Following the discussion in Remark 4, we consider the case where $\mathcal{A}$ is topologically closed. Again, this implies that we can assume without loss of generality that $F_{\mathcal{A}} = Q$ in M. It is readily observed that in this case the fixed-point in (24) collapses to

$$\begin{aligned} \mathsf{Win}(M) &= \nu Z \,.\, \mu Y \,.\, \mathsf{Pre}(Y \cup (F_{\mathcal{G}} \cap Z)), \\ &= \nu Z \,.\, \mu Y \,.\, \mathsf{Pre}(Y) \cup (F_{\mathcal{G}} \cap \mathsf{Pre}(Z)), \end{aligned} \tag{29}$$

where we use the short form $\mathsf{Pre}(T) := \mathsf{Pre}(T, \emptyset)$ for the *unconditional inverse dynamics operator*, i.e., $\mathsf{Pre}(T)$ denotes the set of states for which there exists a control pattern such that the target T is attained by the next transition. It should be noted that (11) and (29) are describing the same fixed-point, and this suggests a strong connection between reactive synthesis and supervisory control for the special case of a topologically closed language $\mathcal{A}$.

3.5. Examples

Example 3. Consider the automaton M depicted in Example 3. We now discuss the computation of $\mathsf{Win}(M)$ for four different cases.

▶ **Case 1**: $F_{\mathcal{A}} = F_{\mathcal{G}} = \{s\}$, $\Sigma_{uc} = \Sigma \setminus \{b\}$: As in Example 2, we consider the inner-most fixed-point in (24) and evaluate it by iteratively computing

$$W^0 = \emptyset \quad \text{and} \quad W^i = \mathsf{Pre}((W^{i-1} \setminus F_{\mathcal{A}}) \cup Y^k \cup (F_{\mathcal{G}} \cap Z^l), X^j \setminus F_{\mathcal{A}}). \tag{30}$$

For the initial state sets $Z^0 = X^0 = Q$ and $Y^0 = \emptyset$ we obtain the sequence

$$\begin{aligned} W^1 &= \mathsf{Pre}(\{s\}, Q \setminus \{s\}) = \{q\} \\ W^2 &= \mathsf{Pre}(\{q\} \cup \{s\}, Q \setminus \{s\}) = \{p, q, r\} \\ W^3 &= \mathsf{Pre}(\{p, q, r\} \cup \{s\}, Q \setminus \{s\}) = Q \\ W^4 &= \mathsf{Pre}(Q, Q \setminus \{s\}) = Q = W_3 \end{aligned}$$

giving $X^1 = Q = X^0$ implying $Y^1 = Q$. With this, we re-evaluate (30) and obtain

$$W^1 = \mathsf{Pre}(Q, Q \setminus \{s\}) = Q$$

giving $X^1 = Q = X^0$ and therefore $Y^2 = Q = Y^1$ implying $Z^1 = Q = Z^0$, i.e., the algorithm terminates with $\mathsf{Win}(M) = Q$. Now recall from (27) that the above iteration implies the ranking

$$\mathsf{rank}(s) = (0,0),\ \mathsf{rank}(q) = (1,1),\ \mathsf{rank}(r) = \mathsf{rank}(p) = (2,1).$$

Hence, the resulting supervisor disables b in q.

▶ **Case 2**: $F_{\mathcal{A}} = F_{\mathcal{G}} = \{s\}$, $\Sigma_{uc} = \Sigma$: It should be noted that the controllability status of event b does not change the fixed-point computation, and hence we similarly obtain $\mathsf{Win}(M) = Q$ in this case. However, as $\Sigma_{uc} = \Sigma$ the resulting supervisor enables all available events and lets the plant decide which transition to choose in q. As we assume that the plant only generate runs which correspond to words in $\mathcal{A}$, we know that it will always eventually transition from q to s, implying that the resulting closed loop behaviour fulfils the guarantee.

▶ **Case 3**: $F_{\mathcal{A}} = \{r, s\}$, $F_{\mathcal{G}} = \{s\}$, $\Sigma_{uc} = \Sigma \setminus \{b\}$: Consider again the inner-most fixed-point in (24) initialized with $Z^0 = X^0 = Q$ and $Y^0 = \emptyset$. Then we obtain exactly the same iteration as in c and the algorithm terminates with $\mathsf{Win}(M) = Q$. Again, the given ranking results in a supervisor disabling b in q. However, now this disabling is strictly necessary, as enabling b in q forever enables the plant to generate an accepting run on $M_{\mathcal{A}}$ which is not in $\mathcal{G}$ by simply alternating between r and q.

▶ **Case 4**: $F_{\mathcal{A}} = \{r, s\}$, $F_{\mathcal{G}} = \{s\}$, $\Sigma_{uc} = \Sigma$: Consider again the inner-most fixed-point in (24) initialized with $Z^0 = X^0 = Q$ and $Y^0 = \emptyset$. Then we obtain

$$W^1 = \mathsf{Pre}(\{s\}, Q \setminus \{r, s\}) = \emptyset = W^0$$

and hence $X^1 = \emptyset$. Re-evaluating (30) for $X^1 = \emptyset$ yields

$$W^1 = \mathsf{Pre}(\{s\}, \emptyset) = \emptyset = W^0$$

and hence $X^2 = \emptyset = X^1$, implying $Y^1 = \emptyset = Y^0$ and therefore $Z^1 = \emptyset$. As $\mathsf{Pre}(\{s\}, Q \setminus \{r, s\}) = \emptyset$, the re-evaluation of (30) remains empty and the algorithm terminates with $\mathsf{Win}(M) = \emptyset$. Hence, Problem 2 does not have a solution in this case. Intuitively, this is due to the fact that $b \in \Sigma_{uc}$ implies that the controller cannot prevent the plant from generating an accepting run on $M_{\mathcal{A}}$ which is not in $\mathcal{G}$ by simply alternating between r and q. ◁

Remark 9. It should be noted that the first two cases discussed in Example 3 have the property that $\mathcal{G}$ and $\mathcal{A}$ coincide and therefore it is easy to see that $\mathcal{G}$ is relatively topologically closed w.r.t. $\mathcal{A}$. This implies that a maximally permissive supervisory controller (resulting in the closed loop $\mathcal{K}^{\uparrow}$ in (22)) exists for this example and it can be computed using the more common synthesis procedures for the supervision of $*$-languages (see the discussion in Remark 6). For case 1 of Example 3 it is actually given by a supervisor enabling all available events. This coincides with the controller computed in case 2.

4. Comparison

This section provides a comparison between the reactive synthesis problem, as introduced in Section 2, and the supervisory control problem, as introduced in Section 3. For both problems, the system that one seeks to synthesize can be interpreted as a causal feedback which is meant to be operated in interaction with its respective environment. However, the problems differ in the interpretation of how the system and the environment interact. For reactive synthesis, the system operates in computation cycles with reading inputs and assigning outputs once per cycle. Thus, the system is driven by some mechanism that triggers the cycle and the input readings. In turn, the system drives its environment by output assignments. This contrasts the common interpretation in supervisory control, where the system passively observes past events to apply a control pattern, while the environment is responsible for the actual execution of transitions. However, these interpretations of the interaction do not show up explicitly either in the formal problem statement or in the algorithms. Thus, we may very well consider a reactive system where computation cycles are triggered by the environment and we may also consider supervisors that effectively apply singleton control patterns to actively execute plant transitions. Thus, regarding causality, the different interpretations of system interaction are irrelevant at this stage.

Using this insight, we demonstrate how one can formally transform the two synthesis problems and their solutions into each other. More specifically, we transform the parameters of a reactive synthesis problem such that they constitute a supervisory control problem and we show how any solution of the latter problem can be transformed back to obtain a solution to the initial reactive synthesis problem. From a practical perspective, we thus can employ the algorithm from supervisory control, Section 3.4, to address reactive synthesis problems. As we shall see, the converse transformation, i.e., to solve a supervisory control problem by means of reactive synthesis procedures, is more involved and requires additional assumptions. Here, we identify special cases, in which both problems are equivalent regarding the existence of solutions in general, and also regarding the regularity of solutions for regular parameters.

4.1. Reactive Synthesis via Supervisory Control.

In this section, we show how a solution to the reactive synthesis problem can be computed using the supervisory controller synthesis method presented in Section 3.4. This is done in three steps. Given a particular instance of the reactive synthesis problem we (i) derive a corresponding supervisory control problem, (ii) compute a solution of the latter in terms of a non-blocking supervisor, and (iii) derive a reactive module that solves the original reactive synthesis problem. Step (ii) is already addressed in 3.4 and we are left to discuss Steps (i) and (iii).

The Corresponding Supervisory Control Problem

Given the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$, we are provided the non-empty and disjoint finite sets U and Y and two ω-languages $\mathcal{A}, \mathcal{G} \subseteq (UY)^\omega$. To construct a corresponding supervisory control problem, a natural choice is to associate the assumption $\mathcal{A}$ with the plant and the guarantee $\mathcal{G}$ with the specification. This implies $\Sigma = U \dot{\cup} Y$ and our remaining choice is that of Σ_{uc}. We let $\Sigma_{\mathrm{uc}} = U$ and,

hence, $Y = \Sigma - \Sigma_{\mathrm{uc}}$, which will be justified below. Having set all parameters, we obtain the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ and assume that we are provided a solution in terms of a non-blocking supervisor $f : \Sigma^* \to \Gamma$ with associated behaviour $\mathcal{L}$. The remaining part of this section discusses how to derive a reactive module that solves the original problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$.

Extracting the Reactive Module

As our first observation, we recall from Lemma 2, that the behaviour $\mathcal{L}$ associated with the supervisor f is topologically closed (SC1) and universally controllable (SC2). In contrast, reactive modules are characterised by (RM1) – (RM3) in Lemma 1, where topological closedness (RM1) matches (SC1) and the locally free input (RM2) is implied by universal controllability (SC2) and $Y = \Sigma - \Sigma_{\mathrm{uc}}$. Thus, to transform $\mathcal{L}$ to qualify as the behaviour of a reactive module, we are left to address that the output locally processes the input (RM3).

At a first stage, we trim f to only enable those controllable events that can actually occur, i.e., we consider $h : \Sigma^* \to \Gamma$ with

$$h(s) := \Sigma_{\mathrm{uc}} \cup \{\, \sigma \in f(s) \,|\, s\sigma \in \mathrm{pfx}\,\mathcal{A} \,\} \tag{31}$$

for all $s \in \Sigma^*$. This is not expected to affect the closed-loop behaviour and, indeed, the supervisor f constructed in Section 3.4 already possesses this property. At a second stage, we ensure that at any instance of time exactly one controllable event is enabled, i.e., we consider $f' : \Sigma^* \to \Gamma$ that satisfies

$$f'(s) = \Sigma_{\mathrm{uc}} \,\dot\cup\, \{\sigma\}\,, \quad \text{where } \sigma \in \Sigma - \Sigma_{\mathrm{uc}} \text{ and, if } h(s) \neq \Sigma_{\mathrm{uc}}, \text{ then } \sigma \in h(s)\,, \tag{32}$$

for all $s \in \Sigma^*$. As a special case, f' can be constructed as a composition $f' = h' \circ h$ where $h' : \Gamma \to \Gamma$ is a static filter such that

$$h'(\gamma) = \Sigma_{\mathrm{uc}} \,\dot\cup\, \{\sigma\}\,, \quad \text{where } \sigma \in \Sigma - \Sigma_{\mathrm{uc}} \text{ and, if } \gamma \neq \Sigma_{\mathrm{uc}}, \text{ then } \sigma \in \gamma\,, \tag{33}$$

for all $\gamma \in \Gamma$. In particular, this example demonstrates that f' can be implemented as a finite Mealy automaton provided that $\mathcal{L}$ is ω-regular. Although this second post-processing stage at instances enables an arbitrarily chosen additional controllable event, it does so only when the plant at hand will not accept any controllable event at all. Thus, the second post-processing stage is expected to restrict the closed-loop behaviour.

Technically, f' is a supervisor and, by Lemma 2, the associated behaviour $\mathcal{L}'$ is non-empty and exhibits (SC1) and (SC2). Referring to the second post-processing stage, we obtain the following additional properties:

$$\forall\, s \in \mathrm{pre}\mathcal{L}'\ \exists\, \sigma \in \Sigma - \Sigma_{\mathrm{uc}} :\ \ s\sigma \in \mathrm{pre}\mathcal{L}'\,, \tag{34}$$

$$\forall\, s \in \mathrm{pre}\mathcal{L}',\, \sigma', \sigma'' \in \Sigma - \Sigma_{\mathrm{uc}} :\ \ s\sigma' \in \mathrm{pre}\mathcal{L}',\ s\sigma'' \in \mathrm{pre}\mathcal{L}'\ \Rightarrow\ \sigma' = \sigma''\,, \tag{35}$$

in support of (RM3).

In a third post-processing step, we intersect $\mathcal{L}'$ with $(UY)^\omega$ in order to enforce alternating inputs and outputs, i.e.,

$$\mathcal{L}'' := \mathcal{L}' \cap (UY)^\omega\,. \tag{36}$$

Although the latter construct will formally invalidate (SC2), it retains (RM2) and it does not affect the closed-loop configuration $\mathcal{A} \cap \mathcal{L}'$ since we have $\mathcal{A} \subseteq (UY)^\omega$.

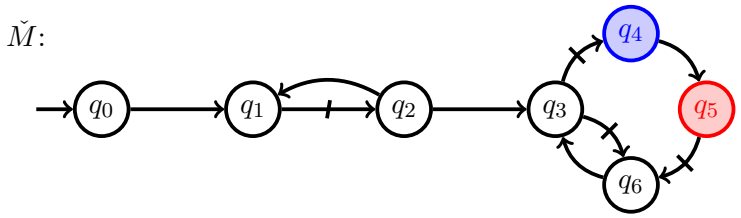

Figure 4: Transition structure of the Büchi automaton $\check{M}$ representing the instance of Problem 2 discussed in Example 4 which corresponds to the instance of Problem 1 represented by M depicted in Figure 2 and discussed in Example 2. The final states $F_{\mathcal{A}} = \{q_4\}$ and $F_{\mathcal{G}} = \{q_5\}$ are indicated in blue and red, respectively. Transitions labelled by controllable events $\Sigma - \Sigma_{uc}$ are indicated by a tick.

Result

We can now state our first main result, i.e., $\mathcal{L}''$ indeed solves the initial instance of the reactive synthesis problem. A proof is given in Section C.

Theorem 2. Given non-empty alphabets U, Y, $U \cap Y = \emptyset$, the assumption $\mathcal{A} \subseteq (UY)^\omega$, and the guarantee $\mathcal{G} \subseteq (UY)^\omega$, consider the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$. Let $\Sigma := U \dot\cup Y$ and $\Sigma_{\mathrm{uc}} := U$. If a supervisor $f : \Sigma^* \to \Gamma$ with associated behaviour $\mathcal{L}$ solves the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$, then $\mathcal{L}''$, as defined by Eqs. (31), (32) and (36), solves $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$. Moreover, $\mathcal{A}$ and $\mathcal{L}''$ are non-conflicting. If $\mathcal{L}$ is ω-regular, then f' can be chosen to be realisable by a finite automaton, and, in turn, $\mathcal{L}''$ is ω-regular. □

By the above theorem, for any reactive synthesis problem under consideration for which the corresponding supervisory control problem exhibits a solution, we can use this solution to construct a reactive module that solves the original reactive synthesis problem. For practical purposes, the overall procedure amounts to the synthesis algorithm from supervisory control, Section 3.4, and the additional post-processing given by Eqs. (31)-(36). This is demonstrated by the following two examples.

Example 4. Consider the reactive synthesis problem discussed in Example 2. There, the automaton M depicted in Figure 2 with $T^0 = \{q_4\}$ and $T^1 = \{q_5\}$ is used to derive a two player game and to compute a winning strategy for the system player in this game which can be translated into a reactive module solving the given instance of Problem 1. We now show how to translate M into an automaton $\check{M}$ conforming with (23) which can be used to solve the corresponding supervisory control problem via Proposition 2.

Due to the assumptions made on $\mathcal{A}$ and $\mathcal{G}$ in Section 2, it is easy to see that $\mathcal{A} = \mathrm{L}_{\mathrm{m}}^{\omega}(M_{\mathcal{A}})$, $\mathcal{G} = \mathrm{L}_{\mathrm{m}}^{\omega}(M_{\mathcal{G}})$ and $\operatorname{pfx}\mathcal{A} = \mathrm{L}^*(M)$. The only structural difference between M and $\check{M}$ is that the former distinguishes environment and system states, while the latter distinguishes controllable (ticked) and uncontrollable transitions. However, due to the alternation of $\Sigma_{uc} = U$ and $\Sigma_c = Y$ in $\mathcal{A}$ and $\mathcal{G}$, these changes are only cosmetic and we essentially get the same automaton as before. For convenience, we have depicted the resulting automaton $\check{M}$ conforming with (23) in Figure 4.

We now synthesize a non-blocking supervisor from the problem representation $\check{M}$ in Figure 4 by evaluating the four-nested fixed-point in (24) for this automaton $\check{M}$.

As in Example 3, we consider the inner-most fixed-point of (24) and evaluate it by iteratively computing

$$W^0 = \emptyset \quad \text{and} \quad W^i = \mathsf{Pre}((W^{i-1} \setminus F_{\mathcal{A}}) \cup Y^k \cup (F_{\mathcal{G}} \cap Z^l), X^j \setminus F_{\mathcal{A}}). \tag{37}$$

For the initial state sets $Z^0 = X^0 = Q$ and $Y^0 = \emptyset$ we obtain the sequence

$$W^1 = \mathsf{Pre}(\{q_5\}, Q \setminus F_{\mathcal{A}}) = \{q_4\}$$
$$W^2 = \mathsf{Pre}(\{q_5\}, Q \setminus F_{\mathcal{A}}) = W^1$$

giving $X^1 = \{q_4\}$. With this, we re-evaluate (37) and obtain

$$W^1 = \mathsf{Pre}(\{q_5\}, \emptyset) = \{q_4\}$$
$$W^2 = \mathsf{Pre}(\{q_5\}, \emptyset) = W^1$$

giving $X^2 = \{q_4\} = X^1$ and therefore $Y^1 = \{q_4\}$. Now we reset X to $X^0 = Q$ and re-evaluate (37), giving

$$W^1 = \mathsf{Pre}(\{q_4, q_5\}, Q \setminus F_{\mathcal{A}}) = \{q_3, q_4\}$$
$$W^2 = \mathsf{Pre}(\{q_3, q_5\}, Q \setminus F_{\mathcal{A}}) = \{q_2, q_3, q_4, q_6\}$$
$$W^3 = \mathsf{Pre}(\{q_2, q_3, q_5, q_6\}, Q \setminus F_{\mathcal{A}}) = \{q_1, \ldots, q_6\}$$
$$W^4 = \mathsf{Pre}(\{q_1, \ldots, q_6\}, Q \setminus F_{\mathcal{A}}) = Q$$

Hence, we obtain $X^1 = Q = X^0$ and therefore $Y^2 = Q$. With this, the re-evaluation of (37) terminates again with $W^\infty = Q$ giving $Y^3 = Q$ and hence $Z^1 = Q = Z^0$, i.e., the algorithm terminates. This results in the ranking

$$\mathsf{rank}(q_5) = (0,0),\ \mathsf{rank}(q_4) = (1,1),\ \mathsf{rank}(q_3) = (1,2),\ \mathsf{rank}(q_2) = \mathsf{rank}(q_6) = (2,2),$$
$$\mathsf{rank}(q_1) = (3,2),\ \mathsf{rank}(q_0) = (4,2).$$

Defining a supervisor based on this ranking and extracting a reactive module via (31)-(36) results in a system strategy which always transitions to q_4 from q_3. ◁

By comparing Example 2 and Example 4, we see that the four-nested fixed point used in Example 4 allows us to distinguish between transitioning from q_3 to q_4 or to q_6 and, as a consequence, choose the former to not falsify the assumptions. This constitutes a more desirable solution to the reactive synthesis problem given by M as depicted in Figure 2, then the one computed via the three-nested fixed point (9) in Example 2.

Remark 10. Similarly to the observation in Remark 9 on solutions of Example 3, it should be noted that for the supervisory control problem discussed in Example 4 we have that $\mathcal{G}$ is relatively topologically closed w.r.t $\mathcal{A}$, and we can therefore compute a maximally permissive supervisor. Again, this supervisor simply enables every available transition in every state and therefore leaves the choice to the plant whether it transitions to q_4 or q_6 in q_3. As we assume that the plant only generates runs which correspond to words in $\mathcal{A}$, we know that it will always eventually transition from q_3 to q_4, implying that the resulting closed loop behaviour fulfils the guarantee.

We have seen that any solution to the corresponding supervisory control problem via Theorem 2 results in a particular solution to the original reactive synthesis problem, namely a reactive module whose associated behaviour does not conflict with

$\mathcal{A}$, i.e., the computed module does not falsify the assumptions. The question of how to synthesize reactive modules which do not falsify the assumptions has recently gained some attention from the reactive synthesis community. We will therefore discuss some recent advances in this direction and relate them to the reactive modules obtained via Theorem 2 in Section 5.

4.2. Supervisory Control via Reactive Synthesis

We now consider the supervisory control problem and aim for a solution in terms of a reactive module. Similarly to our approach in Section 4.1, we transform a particular instance of the supervisory control problem into a corresponding instance of a reactive synthesis problem. To this end, we establish a general transformation to match the input ranges and output ranges of the respective feedback maps such that we can address the behavioural requirements (SC1) and (SC2) by means of (RM1)–(RM3). However, a distinguishing feature of a solution to a supervisory control problem is that the closed-loop configuration must be non-conflicting and this is not explicitly addressed by reactive synthesis. We therefore need to impose additional conditions on the problem parameters to establish a non-conflicting closed loop.

4.2.1. Control-Patterns as System Outputs

In this section, we match the ranges of the respective feedback maps without imposing any a-priori assumptions on the problem parameters. In this sense, our approach here is rather general. However, to obtain a qualifying supervisor, we will need to impose relevant restrictions in retrospect. Similar to Section 4.1, our approach is organised in three steps. Given a particular instance of a supervisory control problem, we (i) derive a corresponding reactive synthesis problem, (ii) identify a solution of the latter in terms of a reactive module, and (iii) derive a supervisor that solves the original control problem.

The Corresponding Reactive Synthesis Problem

Given a supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$, we are provided an alphabet Σ, a set of uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, a plant behaviour $\mathcal{A} \subseteq \Sigma^\omega$ and an upper-bound specification $\mathcal{G} \subseteq \Sigma^\omega$ on the closed-loop behaviour.

As before, we associate the supervisory controller with the reactive module, i.e., the system to be designed. In order to define the input range U and the output range Y, recall from Section 3.1 that a supervisor is a map $f : \Sigma^* \to \Gamma$ that applies a control pattern $\gamma = f(s)$ after the system has generated the sequence $s \in \Sigma^*$. The system in turn generates the next event $\sigma \in \Gamma$, and so forth. Therefore, a nearby choice of U and Y is given by Σ and Γ, respectively.

The interaction of the supervisor and the plant always starts with the former applying a control pattern $\gamma = f(\epsilon)$, i.e., the system to be designed has the first move. In contrast, in our description of reactive synthesis, any run begins with a move by the environment. We therefore introduce a distinguished dummy event $0 \notin \Sigma$ which we will use below to pass on the first move to the system to be designed; i.e.,

$$\Sigma' := \Sigma \cup \{0\}, \quad \Sigma'_{\mathrm{uc}} := \Sigma_{\mathrm{uc}} \cup \{0\}, \qquad \Gamma' := \{\gamma \subseteq \Sigma' \mid \Sigma'_{\mathrm{uc}} \subseteq \gamma\} \tag{38}$$

and define $U := \Sigma'$ and $Y := \Gamma'$. With this choice, a reactive synthesis problem refers to ω-languages that are subsets of $(UY)^\omega = (\Sigma'\Gamma')^\omega$ and we need to transform our problem parameters $\mathcal{A}, \mathcal{G} \subseteq \Sigma^\omega$ accordingly. We begin with the specification $\mathcal{G}$ by pre-pending the distinguished event $0 \in \Sigma'$ and by interleaving any control-patterns between each two events from Σ to obtain

$$\mathcal{G}' := \{\alpha \in (\Sigma'\Gamma')^\omega \mid \mathrm{p}_{\Sigma'}(\alpha) \in 0\mathcal{G}\}\,. \tag{39}$$

The plant $\mathcal{A}$ is transformed similarly, except that we additionally need to encode that once a control pattern $\gamma \in \Gamma'$ has been applied, the next event $\sigma \in \Sigma$ will be within γ. We obtain

$$\mathcal{A}' := \left\{\alpha \in (\Sigma'\Gamma')^\omega \middle| \begin{pmatrix} \mathrm{p}_{\Sigma'}(\alpha) \in 0\mathcal{A} \\ \wedge \forall \gamma \in \Gamma',\, \sigma \in \Sigma' \,.\, s\gamma\sigma \in \mathrm{pfx}\,\alpha \Rightarrow \sigma \in \gamma \end{pmatrix}\right\} \tag{40}$$

Now, we solve the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$ with $U = \Sigma'$, $Y = \Gamma'$, $\mathcal{A}' \subseteq (UY)^\omega$, and $\mathcal{G}' \subseteq (UY)^\omega$ as defined in (38)-(40), in order to obtain a reactive module $r : U^+ \to Y$ with associated behaviour $\mathcal{L}'$.

Extracting the Supervisor

Let $r : U^+ \to Y$ be a solution of the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$ with associated behaviour $\mathcal{L}'$. For the supervisor, consider the candidate behaviour

$$\mathcal{L} := \left\{\beta \in \Sigma^\omega \middle| \exists \alpha \in \mathcal{L}' \,.\, \begin{pmatrix} 0\beta = \mathrm{p}_{\Sigma'}(\alpha) \\ \wedge \forall \gamma \in \Gamma',\, \sigma \in \Sigma' \,.\, s\gamma\sigma \in \mathrm{pfx}\,\alpha \Rightarrow \sigma \in \gamma \end{pmatrix}\right\}. \tag{41}$$

It is shown in Appendix C, Proposition 4, that $\mathcal{L}$ satisfies (SC1), (SC2) and we have that $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$, as consequences of (RM1) and (RM2) holding for $\mathcal{L}'$ and of $\mathcal{L}' \subseteq \mathcal{A}' \to \mathcal{G}'$. In particular, $\mathcal{L}$ is the behaviour associated with a supervisor that enforces the upper bound specification $\mathcal{K} = \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{G}$. Furthermore, it is shown in Appendix C, Proposition 5 that $\mathrm{pfx}\,\mathcal{L}$ and $\mathrm{pfx}\,\mathcal{A}$ do not deadlock. Referring back to Problem 2, we are left to show that $\mathcal{K}$ is non-empty and that $\mathcal{A}$ and $\mathcal{L}$ are non-conflicting, or to give conditions under which this holds.

Result

Whenever $\mathcal{A}$ is topologically closed, we know that the absence of deadlocks in the closed loop, Eq. (4), implies non-conflictingness of $\mathcal{A}$ and $\mathcal{L}$, Eq. (19). Hence we can use topologically closedness as a sufficient condition to conclude that our candidate supervisor solves the control problem.

Theorem 3. Given a finite alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, a plant $\mathcal{A} \subseteq \Sigma^\omega$ and a specification $\mathcal{G} \subseteq \Sigma^\omega$, consider the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$. Preprocess the parameters according to Eqs. (38) to (40) to obtain $U = \Sigma'$, $Y = \Gamma'$ and $\mathcal{A}, \mathcal{G}' \subseteq (UY)^\omega$. Let $\mathcal{L}'$ denote a solution to the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$. If the plant $\mathcal{A}$ is topologically closed, then $\mathcal{L}$ defined by Eq. (41) solves $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$. If $\mathcal{L}'$ is ω-regular, then so is $\mathcal{L}$. □

By the above theorem we have the following result. Given an instance of the supervisory control problem with topologically closed plant and assuming that the corresponding reactive synthesis problem has a solution, we can obtain a non-blocking

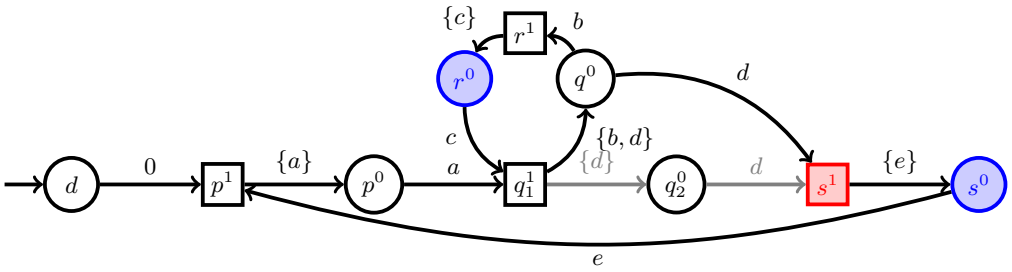

Figure 5: Transition structure of the Büchi automaton $\check{M}$ representing the instance of Problem 1 discussed in Example 5, which corresponds to the instance of Problem 2 represented by M depicted in Figure 3 with $\Sigma_{uc} = \Sigma \setminus \{b\}$ (including state q'^0) and $\Sigma_{uc} = \Sigma$ (excluding state q'^0). The final state sets $T^0 = \{r^0, s^0\}$ and $T^1 = \{s^1\}$ are indicated in blue and red, respectively.

supervisor solving the initial control problem. Technically, the overall procedure amounts to pre-processing by Eqs. (38) to (40), the reactive synthesis procedure discussed in Section 2.3, and post-processing by Eq. (41). This is illustrated by the following example.

Example 5. Recall from Example 3, that M depicted in Figure 3 represents $\mathcal{G}$ and $\mathcal{A}$ containing all strings visiting $F_{\mathcal{G}} = \{s\}$ and $F_{\mathcal{A}} = \{s, r\}$ infinitely often, respectively. As M in Figure 3 cannot generate infinite strings which do not visit either r or s infinately often, it is easy to see that $\mathcal{A}$ is topologically closed. Hence, we can solve this instance of Problem 2 via Theorem 3. To obtain the corresponding instance of Problem 1 w.r.t. $U = \Sigma'$, $Y = \Gamma'$, $\mathcal{A}'$ and $\mathcal{G}'$ (as defined in (38)-(40)), we manipulate M in Figure 3 s.t. the resulting automaton $\check{M}$ conforms with (7) and represents $\mathcal{A}'$ and $\mathcal{G}'$.

This essentially amounts to splitting every state $q \in Q$ of M into a system state q^1 and k environment states q_k^0 with $k \in \{1, \ldots, K\}$ and K being the number of possible control patters available in q. Then the transition from q^1 to q_k^0 is labeled with the respective control pattern $\gamma_k \subseteq \Gamma$. Furthermore, outgoing transitions of q_k^0 mimic outgoing transitions of q in M with the restriction that only transitions labeled with symbols within the k-th control pattern are allowed. Hence, a transition $(q, \sigma, q') \in \delta$ of M is copied to all $(q_k^1, \sigma, q'^0) \in \check{\delta}$ of M for which $\sigma \in \gamma_k$. Finally, we add a dummy initial state d whose outgoing transition is labeled by the dummy event 0 and leads to the system part p^1 of M's initial state p.

The resulting automaton $\check{M}$ is depicted in Figure 5. For simplicity, we have trimmed all γ-labels to the events actually available at its source state. The sets of final states are translated in the obvious way, i.e., we obtain $T^0 = \{r^0, s^0\}$ and $T^1 = \{s^1\}$, indicated in blue and red, respectively. It should be noted that depending on the controllability status of event b we get one or two possible control patterns in state q. I.e., if $b \in \Sigma_{uc}$ we obtain the automaton without q_2^0, while $b \in \Sigma \setminus \Sigma_{uc}$ results in the full automaton containing both q_1^0 and q_2^0. We now use $\check{M}$ as the input to the reactive synthesis algorithm in Section 2.3 and consider these two cases separately.

▶ $\Sigma_{uc} = \Sigma$: In this case the system has no choice in any of the system states. Therefore, it cannot prevent the environment to always take transition b in q^0. Hence, the set of winning states is empty in this case. It should be noted that this coincides with the solution obtained in the forth case of Example 3.

▶ $\Sigma_{uc} = \Sigma \setminus b$: Here the system has a choice in q^1 and can apply the control

pattern $\{d\}$, effectively disabling b in q of M in Figure 3. It should be noted that this coincides with the solution obtained in the third case of Example 3. ◁

Equivalence of Problem Statements

Given the resuts in Theorem 2 and Theorem 3, we have established for topologically closed plants that both synthesis problems can be solved via the respective other one. However, in our construction we assume that the respective target problem exhibits a solution. Since Theorem 3, in contrast to Theorem 2, uses a non-trivial transformation of the problem parameters $\mathcal{A}$ and $\mathcal{G}$, the two theorems alone do not establish equivalence of the two problems regarding solvability. For this purpose, we show in Appendix C, Proposition 6, that if the control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ exhibits a solution, then so does the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$ with parameters defined via Eqs. (39)-(40). In other words, our transformation of the supervisory control problem to an instance of the reactive synthesis problem retains solvability. Referring to Theorem 3, we obtain the following corollary.

Corollary 1. Let Σ be a finite alphabet with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$. For any non-empty topologically closed behaviour $\mathcal{A} \subseteq \Sigma^\omega$ and any upper bound specification $\mathcal{G} \subseteq \Sigma^\omega$, let $U = \Sigma'$, $Y = \Gamma'$, $\mathcal{A}'$ and $\mathcal{G}'$ as defined by Eqs. (38)-(40). Then the control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ has a solution if and only if the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$ has a solution. □

The above corollary matches the close relations ship of the 2-nested fixed-point algorithms (11) and (29) used as a basis to synthesise supervisors and reactive modules for topologically closed plants, respectively.

4.2.2. Alternating Plant Behaviours

The transformation proposed in the previous section turned out technically involved because it needed to encode a mechanism to interleave plant symbols with control patterns in a single language. Therefore, we expect considerable simplifications when turning to the special situation of alternating plant behaviours, i.e., plant behaviours in which controllable and uncontrollable events alternate, and which, in this regard, more closely reassemble the reactive synthesis setting. Technically, we consider the supervisory control problem with parameters $\Sigma_{\mathrm{uc}} \subsetneq \Sigma$ and $\mathcal{A}$, $\mathcal{G} \subseteq (\Sigma_{\mathrm{uc}}(\Sigma - \Sigma_{\mathrm{uc}}))^\omega$. We refer to this class of plants as *input-output behaviours* and we present the corresponding transformation to solve Problem 2 via reactive synthesis. In contrast to the more general case discussed in the previous section, the transformation only affects the solution $\mathcal{L}$ but not the problem parameters $\mathcal{A}$ and $\mathcal{G}$.

The Corresponding Reactive Synthesis Problem

By the alternation of controllable and uncontrollable events, we can choose the correspondence $U := \Sigma_{\mathrm{uc}}$ and $Y := \Sigma - \Sigma_{\mathrm{uc}}$ and obtain $\mathcal{G}$, $\mathcal{A} \subseteq (UY)^\omega$; i.e., our choice constitutes qualifying parameters for reactive synthesis. For the following, we consider a reactive module with behaviour $\mathcal{L}$ that solves the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$ and seek to construct a non-blocking supervisor that solves the original supervisory control problem.

Extracting the Supervisor

The solution $\mathcal{L}$ of the reactive synthesis problem satisfies (RM1)–(RM3) and we need to derive a behaviour that satisfies (SC1) and (SC2) for a solution of the supervisory control problem. Topological closedness (SC1) is immediate by (RM1). Regarding (SC2), we propose the following transformation:

$$\mathcal{L}' := \mathcal{L} \cup ((\mathrm{pfx}\,\mathcal{L})(\Sigma_{\mathrm{uc}}^{\omega}))\,. \tag{42}$$

It is shown in Appendix C, Proposition 7, that the above construct $\mathcal{L}'$ indeed satisfies (SC1) and (SC2) and, moreover, retains the absence of deadlocks, i.e., $\mathrm{pfx}\,\mathcal{L}'$ and $\mathrm{pfx}\,\mathcal{A}$ do not deadlock. For $\mathcal{L}'$ to solve the control problem, we are left to establish that its corresponding supervisor is non-blocking and that it enforces the language inclusion specification; technically, $\mathcal{A}$ and $\mathcal{L}'$ must be non-conflicting with $\emptyset \neq \mathcal{A} \cap \mathcal{L}' \subseteq \mathcal{G}$.

Result

Using the same reasoning as in Section 4.2.1, we see that non-conflictingness of $\mathcal{A}$ and $\mathcal{L}'$ is implied by the absence of deadlocks provided that $\mathcal{A}$ is topologically closed. Using this sufficient condition, we get the following result on the synthesis of supervisors for input-output behaviours via reactive synthesis.

Theorem 4. Given a finite alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, a plant $\mathcal{A} \subseteq \Sigma^{\omega}$ and a specification $\mathcal{G} \subseteq \Sigma^{\omega}$, consider the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$. Assume that the plant exhibits an input-output behaviour, i.e., $\mathcal{A} \subseteq (\Sigma_{\mathrm{uc}}(\Sigma - \Sigma_{\mathrm{uc}}))^{\omega}$, and, without loss of generality, that $\mathcal{G} \subseteq \mathcal{A}$. Let $\mathcal{L}$ denote a solution to the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$, where $U = \Sigma_{\mathrm{uc}}$, $Y = \Sigma - \Sigma_{\mathrm{uc}}$. If the plant $\mathcal{A}$ is topologically closed, then $\mathcal{L}'$ defined by Eq. (42) solves $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$. If $\mathcal{L}'$ is ω-regular, then so is $\mathcal{L}$. □

Given the results in Theorem 2 and Theorem 4, we have established that both synthesis problems can be solved via the respective other one under the assumption of a topologically closed plant. By additionally requiring that $\mathcal{A}$ and $\mathcal{G}$ alternate controllable and uncontrollable events, neither of the proposed transformations affects the problem parameters $\mathcal{A}$ and $\mathcal{G}$. Complementing Corollary 1, we note that our interpretation of the reactive synthesis problem as an instance of the supervisory control problem retains solvability.

Corollary 2. Given an alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subsetneq \Sigma$, let $U = \Sigma_{\mathrm{uc}}$ and $Y = \Sigma - \Sigma_{\mathrm{uc}}$. For any non-empty topologically closed behaviour $\mathcal{A} \subseteq (UY)^{\omega}$ and any upper bound $\mathcal{G} \subseteq \mathcal{A} \subseteq (UY)^{\omega}$, the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ has a solution if and only if the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$ has a solution. □

Theorem 2 and Theorem 4 show that in the special case of input-output behaviours a sound transformation is obtained by simply choosing $U = \Sigma_{\mathrm{uc}}$ and $Y = \Sigma - \Sigma_{\mathrm{uc}}$ and keeping $\mathcal{A}$ and $\mathcal{G}$ unchanged otherwise. This simple transformation reduces on the automaton level to (i) transforming states with outgoing transitions from Σ_{uc} (resp. $\Sigma - \Sigma_{\mathrm{uc}}$) to environment (resp. system) states when transforming $\check{M}$ in (23) to M in (7), or (ii) marking outgoing transitions from an environment (resp. system) state uncontrollable (resp. controllable) when transforming M in (7) to $\check{M}$

in (23). (See Figure 2 and Figure 4 for an example.) Therefore, the application of the unconditioned version $\mathsf{Pre}(T) = \mathsf{Pre}(T, \emptyset)$ of the pre-operator defined in (25) on $\check{M}$ coincides with the application of the controllable pre-operator Pre^1 defined in (8) on M. We have seen in Remark 4 and Remark 8 that for topologically closed plant behaviours $\mathcal{A}$ both synthesis algorithms compute the same 2-nested fixed-point, see Eqs. (11) and (29). This implies that both solution techniques coincide also on the automaton level when considering topologically closed and input-output behaviours.

4.3. Non-Falsifiable Assumptions and Strong Non-Anticipation

By our comparison so far the reactive synthesis problem can be solved via supervisory control. However, the converse transformation in general fails as a non-blocking supervisor by definition requires that $\mathcal{L}$ and $\mathcal{A}$ are non-conflicting and this requirement cannot be expressed by an upper-bound specification in the reactive synthesis problem under consideration. We have seen in Section 4.2 that topological closeness of $\mathcal{A}$ can be used as a sufficient condition to ensure that the solution $\mathcal{L}$ obtained via reactive synthesis is such that $\mathcal{L}$ does not conflict with $\mathcal{A}$. In this section, we present two alternative weaker conditions for a non-conflicting closed loop which were independently developed in either field, and we discuss how they relate. Technically, both conditions address the situation of input-output behaviours and, in this regard, follow up our discussion in Section 4.2.2.

Non-Falsifiable Assumptions in Reactive Synthesis

Referring to the simple transformation discussed in Section 4.2.2, we consider an instance of the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ where $\mathcal{A}, \mathcal{G} \subseteq (\Sigma_{\mathrm{uc}}(\Sigma - \Sigma_{\mathrm{uc}}))^\omega$, and the corresponding reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$, where $U = \Sigma_{\mathrm{uc}}$, $Y = \Sigma - \Sigma_{\mathrm{uc}}$. Provided a solution $\mathcal{L}$ of the reactive synthesis problem, we have that $\mathcal{L}$ and $\mathcal{A}$ do not deadlock, and hence, the closed-loop configuration can continue for infinitely many computation cycles to generate an ω-word $\alpha \in (\mathrm{clo}\,\mathcal{A}) \cap (\mathrm{clo}\,\mathcal{L})$. Since $\mathcal{L}$ is closed, we also have $\alpha \in \mathcal{L}$. However, one may fail on $\alpha \in \mathcal{A}$, and, by the specification $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$, risk that $\alpha \notin \mathcal{G}$. Technically, the problem statement of reactive synthesis does not prevent the construction of a reactive module such that there exists $s \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L})$ but $s \notin \mathrm{pfx}(\mathcal{A} \cap \mathcal{L})$. For such unfortunate words we have for all extension $\beta \in (U \cup Y)^\omega$ that $s\beta \notin \mathrm{clo}(\mathcal{A} \cap \mathcal{L}) = \mathrm{clo}\,\mathcal{G}$, i.e., the generated ω-word not only fails to satisfy the guarantee $\mathcal{G}$ itself but also any safety properties represented by $\mathrm{clo}\,\mathcal{G}$.

This issue can be avoided if the given assumption $\mathcal{A}$ is *non-falsifiable*[10] in the following sense. Given the two player game interpretation used in the algorithmic synthesis of reactive modules, an assumption is called non-falsifiable, if the environment player has a winning strategy in the Büchi game (H, T^0) over the game graph H. In this case, there exists a causal map by which the environment can organise its moves, which ensures that for any infinite play some final environment state $q \in T^0$ is visited infinitely often, regardless of the moves chosen by the reactive module. In this sense, both players win and we have $\alpha \in \mathcal{A} \cap \mathcal{L}$ for any ω-word generated in the closed-loop configuration.

[10] See [Brenguier et al., 2017], Sec. 3, for an illustrative explanation of this phenomenon, called *Win-under-Hype* there.

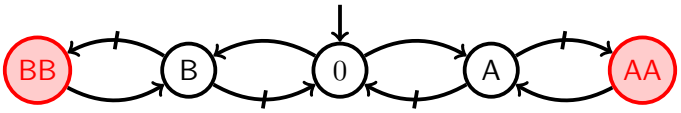

Figure 6: Transition structure of a Büchi-automaton realizing a plant behaviour that needs to anticipate future control patterns in order to satisfy its acceptance condition. Accepting states are marked in red and controllable transitions are indicated by a tick.

Strong Non-Anticipation in Supervisory Control

A closely related issue has been identified in the context of supervisory control in [Moor et al., 2011]. Consider a supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ with plant $\mathcal{A} \subseteq \Sigma^\omega$ and specification $\mathcal{G} \subseteq \mathcal{A}$, and let $\mathcal{L} \subseteq \Sigma^\omega$ denote a solution. As we ask for a non-blocking supervisor, we know that at no specific instance of time the supervisor can prevent the plant to attain its acceptance condition, i.e., for all $s \in K_{\mathrm{loc}}$ we have $s \in \mathrm{pfx}(\mathcal{A} \cap \mathcal{L})$ and there exists $\beta \in \Sigma^\omega$ such that $s\beta \in \mathcal{A} \cap \mathcal{L}$. However, this does not rule out supervisors which require the plant to eventually take certain transitions that depend on future control patterns, i.e., the plant may need to anticipate the moves of the supervisor.

We illustrate this subtle issue by an example adapted from [Moor et al., 2011], given in Figure 6. As states AA and BB are accepting, any supervisor that always eventually enables a transition from A to AA or from B to BB is non-blocking. Whenever in state 0 the plant needs to decide to either take the transition to A or to B. Regardless which decision the plant takes, there is always a non-blocking supervisor that requires just the opposite decision to allow the plant to visit its marking. Only if the plant knew the next control pattern, it could, when in state 0, take a decision that opens the chance for the plant to visit its marking.

In many applications it is unrealistic to assume that the plant knows about future control patterns. Thus, there is an interest in plant behaviours that can attain their acceptance conditions independently of a particular supervisor, and a class of such plant behaviours has been characterized in [Moor et al., 2011] for the special case of input-output behaviours. The results in [Moor et al., 2011] amount to representing $\mathcal{A}$ as a union of topologically closed components that each exhibit $Y = \Sigma - \Sigma_{uc}$ as a locally free input. It is further shown that this condition is equivalent to the controllability prefix of $\mathcal{A}$ w.r.t. the closure $\mathrm{clo}\,\mathcal{A}$ to equal $\mathrm{pfx}\,\mathcal{A}$, i.e.,

$$\mathrm{cfx}_{\mathrm{clo}\,\mathcal{A},\,Y}\,\mathcal{A} = \mathrm{pfx}\,\mathcal{A} \tag{43}$$

where $Y = \Sigma - \Sigma_{uc}$ takes the role the uncontrollable events.

The latter property is referred to as *strong non-anticipation*. Referring to the game theoretic interpretation of supervisory control used in the discussion of the synthesis algorithm in Section 3.4, Eq. (43) requires that the *local plant* $\mathrm{pfx}\,\mathcal{A}$ is always in a winning configuration regarding the satisfaction of its own acceptance condition, i.e., at any time the plant can decide to apply a causal feedback map to choose the next event such that the plant acceptance condition will be met regardless the control imposed by the supervisor. By strong non-anticipation, a non-conflicting closed loop, Eq. (19), is implied by the absence of deadlocks.

Since non-conflictingness imposes its formal requirements only on words within the local closed loop, $(\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L})$, we may alternatively use the weaker condition

$$(\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L}) \subseteq \operatorname{cfx}_{\operatorname{clo}\mathcal{A},Y}\mathcal{A}\,. \tag{44}$$

to conclude a non-conflicting closed loop from the absence of deadlocks; see Appendix C, Proposition 9, for a formal proof. If we synthesise $\mathcal{L}$ by supervisory control, the closed loop will be non-conflicting by construction. However, if we still are interested in establishing Eq. (44), we can pre-process the specification by

$$\mathcal{G}' := \mathcal{G} \cap \{\, \alpha \in \Sigma^\omega \mid \operatorname{pfx}\alpha \subseteq \operatorname{cfx}_{\operatorname{clo}\mathcal{A},Y}\mathcal{A} \,\}\,. \tag{45}$$

With this specification, the synthesis algorithm of supervisory control assures that the local closed-loop evolves within the controllability prefix as required by Eq. (44).

Comparison

When comparing the game theoretic interpretations of non-falsifiable assumption and strong non-anticipation, the former requires the "environment to play clever" from the very beginning, whereas the latter allows the plant to start doing so eventually. This suggests that an assumption is non-falsifiable if and only if $\epsilon \in \operatorname{cfx}_{\operatorname{clo}\mathcal{A},Y}(\mathcal{A})$, which we prove in Appendix C, Proposition 8. Hence, it is easy to see that the condition of a non-falsifiable assumption is weaker then the condition of a strongly non-anticipating plant behaviour. In particular, we have that (i) topological closedness of $\mathcal{A}$ implies (ii) strong non-anticipation, which in turn implies (iii) Eq. (44), to finally imply (iv) $\mathcal{A}$ to be a non-falsifiable assumption. The converse implications, however, do not hold in general. It should be noted that, in contrast to strong non-anticipation, the condition in Eq. (44) constitutes a closed-loop property. As such, we can verify whether Eq. (44) is satisfied after we have obtained $\mathcal{L}$ via reactive synthesis and, if the test passes, conclude that the absence of deadlocks, Eq. (4), implies a non-conflicting closed loop, Eq. (19).

Result

Given the above discussion we can use (44) as a sufficient condition to establish a solution of a supervisory control problem via reactive synthesis analogously to Theorem 4. However, (44) is weaker then topological closeness of $\mathcal{A}$, and, thus, we obtain a generalisation of Theorem 4.

Theorem 5. Given a finite alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\text{uc}} \subseteq \Sigma$, a plant $\mathcal{A} \subseteq \Sigma^\omega$ and a specification $\mathcal{G} \subseteq \Sigma^\omega$, consider the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$. Assume that the plant exhibits an input-output behaviour, i.e., $\mathcal{A} \subseteq (\Sigma_{\text{uc}}(\Sigma - \Sigma_{\text{uc}}))^\omega$, and, without loss of generality, that $\mathcal{G} \subseteq \mathcal{A}$. Let $\mathcal{L}$ denote a solution to the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$, where $U = \Sigma_{\text{uc}}$, $Y = \Sigma - \Sigma_{\text{uc}}$. If (44) holds for $\mathcal{A}$ and $\mathcal{L}$, then $\mathcal{L}'$ defined by Eq. (42) solves $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$. If $\mathcal{L}'$ is ω-regular, then so is $\mathcal{L}$. □

Again referring to Theorem 2 from Section 4.1, but now using Theorem 5 rather than Theorem 4, we obtain the following generalisation of Corollary 2.

Corollary 3. Given an alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\text{uc}} \subsetneq \Sigma$, let $U = \Sigma_{\text{uc}}$ and $Y = \Sigma - \Sigma_{\text{uc}}$. For any non-empty behaviours $\mathcal{A}$, $\mathcal{G} \subseteq$

$(UY)^\omega$ where $\mathcal{A}$ is strongly non-anticipating, the control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ has a solution if and only if the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$ has a solution. □

Referring back to the discussion below Corollary 2, we still have that the automata realizations of the two corresponding problem instances are directly connected and induce an equivalence of the respectively applied pre-operators. However, as $\mathcal{A}$ is not assumed to be topologically closed, we now have to use the three-nested fixed-point algorithm in (9) to solve the synthesis problem. Nevertheless, this shows that for strongly non-anticipating plants, supervisory controller synthesis reduces to the evaluation of a 3-nested fixed-point.

Example 6. Consider the supervisory control problem represented by $\check{M}$ depicted in Figure 4 and the automaton M depicted in Figure 2 representing the corresponding instance of Problem 1. Recall from Example 2 that a solution of this instance of Problem 1 is given by a reactive module which always transitions from from q_3 to q_6. The only possible run on M is hence given by the sequence $q_0q_1q_2(q_3q_6)^\omega$, and this run generates an ω-word $\alpha \in (\operatorname{clo}\mathcal{A}) \cap \mathcal{L}$ but $\alpha \notin \mathcal{A}$. In particular, we have $\operatorname{pfx}\alpha \subseteq (\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L})$ and, if our condition in Eq. (44) was satisfied, we had that $\operatorname{pfx}\alpha \subseteq \operatorname{cfx}_{\operatorname{clo}\mathcal{A},Y}\mathcal{A}$. Here, the definition of the controllability prefix, Definition 2, requires that there exists a choice of uncontrollable events, by which the plant can enforce to attain its acceptance condition. However, this is not possible given that the choice of controllable events taken by the module only allows for the run $q_0q_1q_2(q_3q_6)^\omega$. Therefore, our condition Eq. (44) is not satisfied for this example. This is as expected, since the given solution to Problem 1 actively falsifies the assumptions. ◁

5. Discussion

So far, we have formally compared supervisory controller synthesis for non-terminating processes and a version of reactive synthesis which incorporates environment assumptions. As one of our main results we have shown that one can solve the considered reactive synthesis problem using supervisory controller synthesis to obtain a reactive module which will not falsify the assumptions on the environment (see Theorem 2). The question of how to synthesize a reactive module which does not falsify the assumptions has recently gained some attention from the reactive synthesis community. This section compares some recent advances in this direction with our solution based on the SCT perspective. We will base our informal discussion mostly on illustrative examples and intuitive explanations.

Cooperative Reactive Synthesis Cooperative Reactive Synthesis (CRS) was introduced in [Bloem et al., 2015] and solves a synthesis problem which is closely related to Problem 1 but substitutes the single specification $\mathcal{A} \rightarrow \mathcal{G}$ in (5) with a *set* of different logical combinations of $\mathcal{A}$ and $\mathcal{G}$, organized in a hierarchy of cooperation levels. The highest element in this hierarchy of specifications is $\mathcal{A} \cap \mathcal{G}$, as the most desirable situation is to ensure that both $\mathcal{A}$ and $\mathcal{G}$ hold. However, this might not always be realizable. If so, the authors propose a systematic way to relax this specification until a solution to the synthesis problem can be found. When going down in the hierarchy to do so, the authors favour the satisfaction of $\mathcal{G}$ over the satisfaction of $\mathcal{A}$. Hence, solutions to synthesis problems along that hierarchy might sacrifices $\mathcal{A}$ in favour of enforcing $\mathcal{G}$.

The solution to this synthesis problem is computed by solving an extended version of the "usual" reactive synthesis algorithm involving the translation of a specification given in LTL to deterministic Rabin automata. The main feature of the algorithm is that it allows to switch to higher cooperation levels during synthesis, when a specification from a higher level becomes satisfiable during the game. This allows to synthesize *maximally cooperative strategies* in the sense that the system makes as much effort as possible to fulfill both $\mathcal{A}$ and $\mathcal{G}$.

Algorithmically, our solution via SCT differers from this approach in that the additional property ensured by SCT, namely non-conflictingness, cannot be expressed by an LTL formula. Hence, the four-nested fixed-point algorithm used to solve a given RS problem via SCT is not the translation of an LTL synthesis problem into a μ-calculus formula. As a result, solutions to both synthesis problems differ in the way assumptions are handled. For example, if the specification $\mathcal{A} \cap \mathcal{G}$ is realizable in the CRS setting, this means that there exists a system strategy that enforces *both* $\mathcal{A}$ and $\mathcal{G}$. On the contrary, a supervisor computed for this problem can assume that $\mathcal{A}$ will always be satisfied by the environment as long as the system is not preventing it from doing so. Hence, a system strategy extracted from this supervisor does not necessarily *enforce* $\mathcal{A}$ but only makes sure that it always remains satisfiable. We illustrate this difference by the following example.

Example 7. Consider the automation M depicted in Figure 7 (left) as representing the instances of Problem 1 we are interested in. The input automaton to the corresponding supervisory synthesis problem[11] is given by $\check{M}$ depicted in Figure 7 (left). We consider two different cases w.r.t. the sets $F_{\mathcal{A}} = T^0$ and $F_{\mathcal{G}} = T^1$. For each case

[11] See Example 4 for a discussion of the conversion.

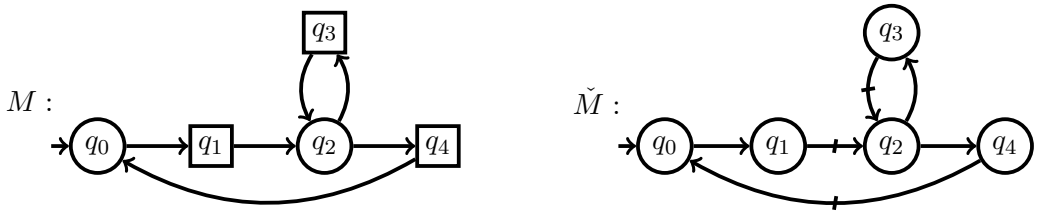

Figure 7: Transition structure of the automaton M (right) representing an instance of Problem 1 considered in Example 7 and the automaton $\check{M}$ (left) representing the corresponding instance of Problem 2.

we discuss the solution obtained by: (RS) – the reactive synthesis algorithm applied to M (see Section 2.3), (SC) – the supervisory controller synthesis algorithm applied to $\check{M}$ (see Section 3.4 and Section 4.1), and (CRS) – the Cooperative reactive synthesis algorithm from [Bloem et al., 2015].

▶ **Case 1**: $T^0 = F_{\mathcal{A}} = \{q_0\}$ and $T^1 = F_{\mathcal{G}} = \{q_4\}$:

▷ **RS**: Running the fixed-point algorithm in (9) on M returns the whole state set as the winning set. As there is essentially no choice for the system player in this game, we see that all transitions in the automaton stay enabled. The interpretation of this solution is that no matter which edge the environment chooses in q_2, it will either falsify the assumption or fulfill both the assumption and the guarantee and is hence a suitable solution in both cases.

▷ **SC**: The transition structure of $\check{M}$ results in a similar synthesis problem as the one discussed in case 2 of Example 3; the resulting supervisor enables any controllable edge. I.e., the reactive module extracted from this supervisor coincides with the one computed via RS. However, the interpretation of its soundness differs; given that $F_{\mathcal{A}} = \{q_0\}$, it is known that the environment chooses to transition from q_2 to q_4 infinitely often, and hence, the guarantee will always be fulfilled.

▷ **CRS**: First note that $\mathcal{G} \cap \mathcal{A}$ cannot be enforced, as the transition from q_2 to q_3 cannot be disabled. The highest realizable specification in the cooperation hierarchy of CRS is given by $G\langle E\rangle(\mathcal{G} \cap \mathcal{A})$ denoting that there always exists a continuation of the current trace which satisfies $\mathcal{G} \cap \mathcal{A}$. The strategy that enforces this is equivalent to the RS and the SC strategy discussed before.

▶ **Case 2**: $T^0 = F_{\mathcal{A}} = \{q_0, q_3\}$ and $T^1 = F_{\mathcal{G}} = \{q_4\}$:

▷ **RS**: Running the fixed-point algorithm in (9) on M returns the empty set, as there is no way to prevent the system from entering state q_2 and preventing the environment to always transition to q_3 from there.

▷ **SC**: Due to the same reason, there also exists no supervisor solving this synthesis problem.

▷ **CRS**: Still $\mathcal{G} \cap \mathcal{A}$ cannot be enforced. The highest realizable specification in the cooperation hierarchy of CRS is given by $\mathcal{A} \cap G\langle E\rangle\mathcal{G}$ denoting that any trace will satisfy $\mathcal{A}$ and there always exists a continuation of the current trace which satisfies $\mathcal{G}$. The strategy that enforces this is equivalent the one computed in case 1. ◁

Interestingly, the CRS solutions coincide in both cases; *hoping* that the plant cooperates s.t. $\mathcal{A} \cap \mathcal{G}$ will be fulfilled. By viewing the assumptions $\mathcal{A}$ as a model of the plant, we are *sure* that the plant will cooperate s.t. $\mathcal{A}$ is satisfied. As this is the cooperation needed in case 1 to obtain $\mathcal{A} \cap \mathcal{G}$, the SCT solution coincides with the

CRS solution in this case. As case 5 allows the plant to fulfill $\mathcal{A}$ without ensuring $\mathcal{A} \cap \mathcal{G}$, we obtain an empty SCT solution for case 2, while the CRS solution remains unchanged. Intuitively, the CRS solution hopes for any type of cooperation while SCT uses a particular cooperation type to verify its solutions.

Intuitively, the difference between CRS and SCT steams from the fact that the SCT algorithm interprets the environment as a separate process with an individual strategy while CRS does not assume a particular rational behaviour of the environment w.r.t. the assumptions. The multi-process interpretation of the problem used in supervisory control is also employed in various other extensions of reactive synthesis, including obliging games [Chatterjee et al., 2010], assume-guarantee synthesis [Chatterjee and Henzinger, 2007] or assume-admissible synthesis [Brenguier et al., 2017].

These works compute a strategy for every player in the game w.r.t. its local specification. Translated to our setting, this implies that one computes an environment strategy w.r.t. its local specification given by $\mathcal{A}$ and a system strategy w.r.t. its local specification given by $\mathcal{G}$. These two specifications are usually synchronized; the play is only won if both player stick to their strategies. This view, however, does not directly match the setting of supervisory control where only the system strategy is computed. The environment is simply assumed to behave in accordance with its model, i.e., only generating plays that are in $\mathcal{A}$, but it cannot be directly influenced s.t. a particular strategy can be deployed. This setting used in supervisory control is met most closely by assume-admissible synthesis (AAS), when interpreting its solution slightly differently.

Assume-Admissible Synthesis (AAS) AAS [Brenguier et al., 2017] is based on the concept of strategy dominance. A system strategy f is dominated by another system strategy f' if (i) for all environment strategies g holds that whenever a play π compliant with f and g is winning, then so is the play π' compliant with f' and g, and (ii) there exists an environment strategy g' s.t. the play π' compliant with f' and g' is winning while the play π compliant with f and g' is not. Every strategy that is not dominated is called *admissible*. If a player is restricted to play only admissible strategies, this implies that for every state in the game graph for which this player has a winning strategy, this strategy needs to be executed. In particular, this implies that no moves from winning states to non-winning states are allowed.

Intuitively, solving AAS for two players (the system and the environment) results in two individual synthesis problems, each synthesizing an admissible strategy for one player assuming that the other one is restricted to play only *admissible* strategies. Given that these two games have a solution, we have the following connection to the SCT synthesis procedure. If we only implement the obtained admissible system strategy, we know that (due to the existence of a solution to the AAS problem for the environment player) the environment can still fulfill its own objective. So, similarly to supervisory control, an assume-admissible system strategy assumes a rational behaviour of the environment w.r.t. the assumptions $\mathcal{A}$ even without implementing a particular environment strategy. However, the interpretation of a "rationally behaving other player" differs. We illustrate this difference by the following example.

Example 8. We first consider the instance of an reactive synthesis problem represented by the automaton M depicted in Figure 8 (top left) and the corresponding instance of Problem 2 represented by $\check{M}$ depicted in Figure 8 (bottom left). We

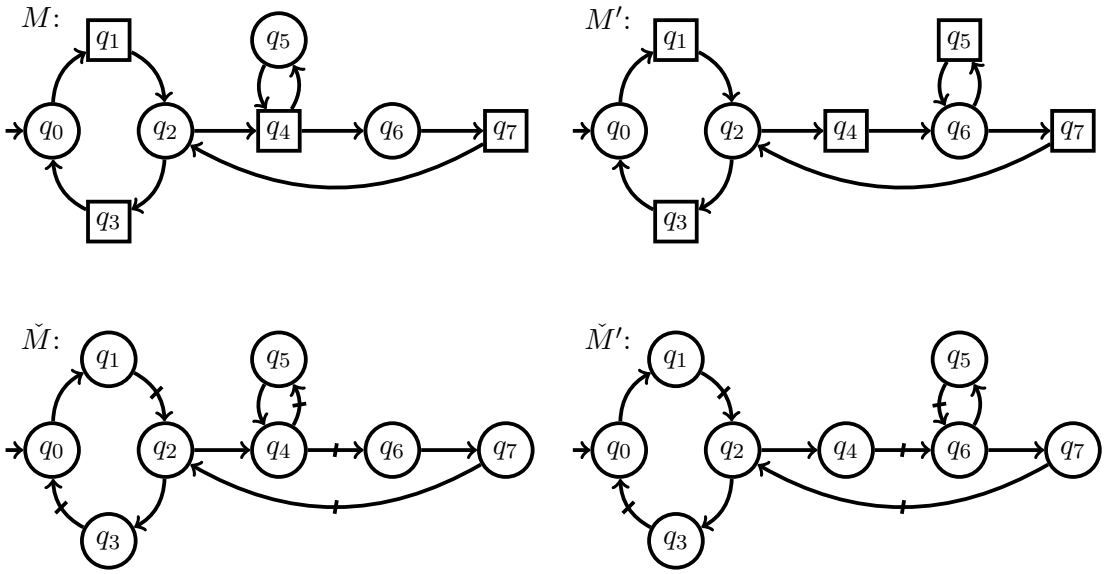

Figure 8: Transition structure of the automata M (resp. M') (top) representing the instance of Problem 1 considered in Example 7 and the automaton $\check{M}$ (resp. $\check{M}'$) (bottom) representing the corresponding instance of Problem 2.

will discuss how assume-admissible strategies (AAS) and supervisors (SC) are constructed for this example using three different cases.

▶ **Case 1a**: M with $T^0 = \{q_0, q_6\}$ and $T^1 = \{q_3\}$:

▷ **SC**: Running the fixed-point in (24) on $\check{M}$ returns the empty set. Hence, this problem has no solution. Intuitively, this is due to the fact that the supervisor cannot prevent the plant from staying in the right component of $\check{M}$, and therefore visiting q_6 infinitely often, but not q_3.

▷ **AAS**: When computing the system strategy, it is assumed that the environment player (player 0) plays an admissible strategy. This means that for any state from which there is a player 0 winning strategy w.r.t. its final states $T^0 = \{q_0, q_6\}$, it will execute exactly this one. It can be observed from M in Figure 8 that player 0 has a winning strategy in state q_0. This strategy is given by always transition from q_2 to q_3. It should be noted that transitioning from q_2 to q_4 instead, allows player 0 to win only if player 1 cooperates (by choosing the transition from q_4 to q_6 infinitely often). I.e., q_4 is not a winning state for player 0, while state q_2 is. Moving from a winning to a maybe-winning state is not part of an assume-admissible strategy.
Based on this observation, the system (player 1) strategy is computed by assuming that player 0 will always transition from q_2 to q_3 and never to q_4. In this case the system strategy is given by always transitioning from q_4 to q_6 (as this is a winning strategy for player 1 and it needs to try to win even if the environment does not play admissible) and choosing the only available transition in every other system state.

▶ **Case 2a**: M with $T^0 = \{q_0, q_6\}$ and $T^1 = \{q_3, q_7\}$:

▷ **SC**: A supervisor obtained via the algorithm in Section 3.4 applied to $\check{M}$ enables all controllable transitions except for the one from q_4 to q_5.

▷ **AAS**: In this case both the system and the environment player have a winning strategy. The resulting strategy profile is however still equivalent to case 1a as the environment has to execute its winning strategy in q_2. Hence, the environment keeps

the play in the left part of the game graph.

▶ **Case 3a**: M with $T^0 = \{q_0, q_6\}$ and $T^1 = \{q_7\}$: :
▷ **SC**: Running the fixed-point in (24) on $\check{M}$ returns the empty set. Hence, this problem has no solution. The reason is similar to case 1a.
▷ **AAS**: As discussed in case 1a, the only admissible strategy of player 0 is to always transition to q_3 from q_2. As this prevents player 1 from reaching its winning state q_7, there exists no assume-admissible player 1 strategy.

▶ **Case 4a**: M with $T^0 = \{q_6\}$ and $T^1 = \{q_3, q_7\}$:
▷ **SC**: We see that $\mathcal{G} \not\subseteq \mathcal{A}$ in this case. We therefore have to use a different marking on $\check{M}$, namely $F_{\mathcal{A}} = \{q_6\}$ and $F_{\mathcal{G}} = \{q_7\}$. Running the synthesis algorithm from Section 3.4 on this automaton returns the same supervisor as in case 2a, as the supervisor can rely on the plant to transition from q_2 to q_4 infinitely often.
▷ **AAS**: Player 0 has no winning strategy in this case. However, admissible strategies are defined such that player 0 will always try its best to still win if player 1 cooperates, hence it will always choose to transition from q_2 to q_4. Given this admissible strategy, player 1 wins if it also tries its best to cooperate and hence always transitions from q_4 to q_6. Hence, AAS results in the same system strategy as SC.

Now consider the automation M' depicted in Figure 8 (top right) as representing the instances of Problem 1 we are interested in. We again consider the four cases discussed for M.

▶ **Case 1b**: M' with $T^0 = \{q_0, q_6\}$ and $T^1 = \{q_3\}$:
▷ **SC**: Running the fixed-point in (24) on $\check{M}'$ returns the empty set due to the same reason as discussed in case 1a.
▷ **AAS**: The change of M to M' causes every state to be winning for player 0. This implies that there exists no assume-admissible system strategy, as the reactive module cannot assume anymore that the environment will transition from q_2 to q_3.

▶ **Case 2b**: M' with $T^0 = \{q_0, q_6\}$ and $T^1 = \{q_3, q_7\}$:
▷ **SC**: The supervisor obtained via the algorithm in Section 3.4 enables every available transition.
▷ **AAS**: In this case both the system and the environment player have a winning strategy. Interestingly, the resulting closed loop behavior is now given by the language accepted by the full automaton, while in case 2a, player 0 is assumed to always keep the play in the left part of M.

▶ **Case 3**: M' with $T^0 = \{q_0, q_6\}$ and $T^1 = \{q_7\}$: :
▷ **SC**: Running the fixed-point in (24) on $\check{M}'$ returns the empty set. Hence, this problem has no solution. The reason is similar to case 1b.
▷ **AAS**: As discussed in case 1b, the admissible strategy of player 0 can transition from q_3 to either q_2 and q_4. This implies that player 0 can prevent player 1 from reaching its winning state q_7 by always choosing to transition from q_3 to q_2. Therefore, there exists no assume-admissible player 1 strategy.

▶ **Case 4b**: M' with $T^0 = \{q_6\}$ and $T^1 = \{q_3, q_7\}$:
▷ **SC**: As in case 4a, we need to consider $\check{M}'$ with marking $F_{\mathcal{A}} = \{q_6\}$ and $F_{\mathcal{G}} = \{q_7\}$. Running the synthesis algorithm from Section 3.4 on this automaton returns the same supervisor as in case 2b, as the supervisor can rely on the plant to transition from q_2 to q_4 infinitely often. Hence, the resulting supervisor simply enables every available transition.
▷ **AAS**: Observe that player 0 has a winning strategy in this case by always transitioning from q_2 to q_4 and from q_6 to q_7. Under this assumption there also exists a player 1 strategy, which is simply given by enabling all available transitions. ◁

Example 8 suggests that the assumptions on the environment player, i.e., on the plant behaviour, are strictly weaker in AAS than in SCT. Hence, there exist situations where AAS has a solution, while there exists no supervisor solving the corresponding synthesis problem (see case 1a). Furthermore, we see that in all cases where a supervisor exists, the assume-admissible system strategy is a subset of (or equivalent to) the reactive module extracted from the computed supervisor. We were not able to construct a case in Example 8 which allows for a supervisor but not for an assume-admissible system strategy. Intuitively, this is due to the fact that the algorithm for supervisory controller synthesis requires $\mathcal{G} \subseteq \mathcal{A}$. Restricting the guarantee for supervisor synthesis to $\mathcal{A} \cap \mathcal{G}$ (case 4a and 4b) rules out all potentially interesting scenarios in this respect. While the above observations conform with our intuition, we have not proven the described connection formally.

A Connection to Probabilistic Games The use of environment assumptions in reactive synthesis makes synthesis problems easier to solve. Hence, a natural question to ask is whether there exists an assumption $\mathcal{A}$ which renders a given (unconstrained) synthesis problem solvable. Ideally, one would like to compute $\mathcal{A}$ s.t. it is "maximally permissive" in the sense that it restricts $\mathcal{G}$ as little as possible and such that it cannot be falsified by a reactive module (in the sense of Section 4.3).

This problem has been addressed by [Chatterjee et al., 2008]. The authors show that computing such maximally permissive assumptions is NP hard and their actual implementation only obtains locally optimal solutions. Interestingly, the considered problem is reduce to finding almost sure winning strategies in probabilistic games.

Indeed, there is a strong connection between the two-nested fixed-point used to calculate almost sure winning strategies in concurrent reachability games [de Alfaro and Henzinger, 2000, de Alfaro et al., 2007] and the two inner-most fixed points of the synthesis algorithm for supervisors, Eq. (24). Both use a pre-operator with two arguments (see Eq. (25)), where the first argument specifies the set which should be reached and the second argument specifies the set which should not be left (resp. surely avoided) while doing so. Intuitively, this connection is not surprising as the setting of supervisory control still leaves the freedom to the plant to choose one of the events allowed by the supervisor. This can also be formalized by a probability distribution over all events available in a given state.

However, the match between these two fixed-points is not exact. Intuitively, the pre-operator used in the supervisory controller synthesis algorithm has the additional twist that source states of valid transitions might be excluded from the winning set. This is required for the outer two fixed-points in Eq. (24) to work properly. It is the subject of further research to see if Eq. (24) can be transformed into a version using a probabilistic pre-operator instead.

6. Conclusion

We have described a variant of reactive synthesis (RS) with upper-bound language-inclusion specification which explicitly addresses environment assumptions and, thus, is a promising candidate when aiming for a comparison with supervisory control theory (SCT). For SCT, we have presented a variant that uses ω-languages as the base model and, hence, match this characteristic feature of RS. For both domains we present technical problem statements and we derive behavioural characterisations of reactive modules and supervisors, respectively. This facilitates a technical comparison. In our attempt to transform problem instances from one domain of research to the other, we make the following core observations.

We succeed unconditionally in solving any instance of the considered RS problem by using SCT; see Section 4.1 (Theorem 2). On the practical side, our proposed transformation retains regularity and we may apply the four-nested fixed-point algorithm from SCT to obtain a reactive module. Notably, the obtained reactive module will not actively falsify the assumptions on the environment. This additional property is enforced by the SCT algorithm as it only computes solutions which do not conflict with the environment assumptions. As the latter property cannot be encoded as an ω-regular property, the four-nested fixed-point algorithm cannot result from a straightforward translation of an LTL synthesis problem into a μ-calculus formula over a two-player game. This provides a new perspective on algorithms ensuring solutions which do not falsify the assumptions, which is an active field of research in the RS community.

The reverse transformation does not work out in general. The reason is that the additional property of non-conflicting solutions required by SCT is not guaranteed by solutions to the RS problem. For this reason, we can only solve a synthesis problem from SCT by RS, if we ensure a non-conflicting closed-loop by imposing additional restrictions on the problem parameters. To this end, we identify three cases: topologically closed plants, Theorem 3, topologically closed plants with alternating inputs and outputs, Theorem 4, and strongly non-anticipating plants with alternating inputs and outputs, Theorem 5. The last, in our opinion, is of particular interest since the additional assumption of strong non-anticipation is weaker then topologically closedness and well motivated for hybrid systems or abstractions thereof, see [Moor et al., 2011]. It is furthermore both conceptionally and technically closely related to non-falsifiable assumptions, a condition developed independently in the RS community. Again, each of the proposed transformations retains regularity of the problem parameters and, hence, can be used to practically synthesize supervisors by algorithms from RS. Hence, for the considered class of plant behaviours we see that a three nested fixed-point computation suffices and may therefore expect computational benefits as a trade-off when imposing additional conditions on the supervisor synthesis problem.

Referring back to the transformation of RS problems via Theorem 2 into synthesis problems in SCT, it is observed that it affects the solutions but not the problem parameters. The same is true for the reverse transformations of input/output behaviours in Theorems 4 and 5. This establishes equivalence of the two problems regarding solvability for respective subclasses of plants as stated in Corollaries 2 and 3. Likewise, we establish by Corollary 1 the equivalence of the transformed problems regarding solvability for topologically closed but non-alternating plants behaviours. Moreover, we show that for topologically closed plant behaviours with

alternating inputs and outputs both fixed-point algorithms collapse to the same 2-nested fixed-point. Using Corollary 2, this establishes equivalence of both synthesis algorithms in this case.

Acknowledgments

We thank Nir Piterman for insightful discussions of the supervisory controller synthesis algorithm and its direct proof contained in Section B.

References

[Bloem et al., 2014] Bloem, R., Ehlers, R., Jacobs, S., and Könighofer, R. (2014). How to handle assumptions in synthesis. In *SYNT'14, Vienna, Austria*, pages 34–50.

[Bloem et al., 2015] Bloem, R., Ehlers, R., and Könighofer, R. (2015). Cooperative reactive synthesis. In *ATVA 2015, Shanghai, China*, pages 394–410.

[Bloem et al., 2012] 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.

[Bourdon et al., 2005] Bourdon, E., Lawford, M., and Wonham, W. M. (2005). Robust nonblocking supervisory control of discrete-event systems. *IEEE Transactions on Automatic Control*, 50:2015–2021.

[Bradfield and Stirling, 2006] Bradfield, J. and Stirling, C. (2006). Modal mu-calculi. In *The Handbook of Modal Logic*, pages 721—756. Elsevier.

[Brenguier et al., 2017] Brenguier, R., Raskin, J.-F., and Sankur, O. (2017). Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83.

[Büchi and Landweber, 1969] Büchi, J. R. and Landweber, L. H. (1969). Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:367–378.

[Cai et al., 2015] Cai, K., Zhang, R., and Wonham, W. M. (2015). Relative observability of discrete-event systems and its supremal sublanguages. *IEEE Transactions on Automatic Control*, 60(3):659–670.

[Chatterjee et al., 2008] Chatterjee, K., Henzinger, T., and Jobstmann, B. (2008). Environment assumptions for synthesis. In *CONCUR*, pages 147–161.

[Chatterjee and Henzinger, 2007] Chatterjee, K. and Henzinger, T. A. (2007). Assume-guarantee synthesis. In *TACAS 2007, Braga, Portugal*, pages 261–275.

[Chatterjee and Henzinger, 2012] Chatterjee, K. and Henzinger, T. A. (2012). A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413.

[Chatterjee et al., 2010] Chatterjee, K., Horn, F., and Löding, C. (2010). Obliging games. In *CONCUR'10, Paris, France*, pages 284–296.

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

[Cury and Krogh, 1999] Cury, J. and Krogh, B. (1999). Robustness of supervisors for discrete-event systems. *IEEE Transactions on Automatic Control*, 44:376–379.

[de Alfaro and Henzinger, 2000] de Alfaro, L. and Henzinger, T. A. (2000). Concurrent omega-regular games. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*, pages 141–154.

[de Alfaro et al., 2007] de Alfaro, L., Henzinger, T. A., and Kupferman, O. (2007). Concurrent reachability games. *Theoretical Computer Science*, 386(3):188 – 217.

[de Querioz and Cury, 2000] de Querioz, M. H. and Cury, J. E. R. (2000). Modular supervisory control of large scale discrete event systems. *WODES*.

[Ehlers et al., 2017] 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.

[Emerson and Jutla, 1991] Emerson, E. and Jutla, C. (1991). Tree automata, mu-calculus and determinacy. In *FOCS'91*, pages 368–377.

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

[Gurevich and Harrington, 1982] Gurevich, Y. and Harrington, L. (1982). Trees, automata, and games. In *STOC '82,San Francisco, USA*, pages 60–65.

[Hopcroft and Ullman, 1979] Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading.

[Klein et al., 2015] Klein, J., Baier, C., and Klüppelholz, S. (2015). Compositional construction of most general controllers. *Acta Informatica*, 52(4):443–482.

[Kozen, 1983] Kozen, D. (1983). Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354.

[Kupferman and Vardi, 2000] Kupferman, O. and Vardi, M. Y. (2000). Synthesis with incomplete information. *In Advances in Temporal Logic*, pages 109–127.

[Kučera, 2011] Kučera, V. (2011). A method to teach the parameterization of all stabilizing controllers. *IFAC Proceedings Volumes*, 44(1):6355 – 6360. 18th IFAC World Congress.

[Lin and Wonham, 1988] Lin, F. and Wonham, W. M. (1988). On observability of discrete-event systems. *Information Sciences*, 44:173–198.

[Maler et al., 1995] Maler, O., Pnueli, A., and Sifakis, J. (1995). *On the synthesis of discrete controllers for timed systems*, pages 229–242.

[Moor, 2016] Moor, T. (2016). A discussion of fault-tolerant supervisory control in terms of formal languages. *Annual Reviews in Control*, 41:159 – 169.

[Moor, 2017] Moor, T. (2017). Supervisory control on non-terminating processes: An interpretation of liveness properties. Technical report, Lehrstuhl für Regelungstechnik, Friedrich-Alexander Universität Erlangen-Nürnberg.

[Moor et al., 2012] Moor, T., Baier, C., Yoo, T.-S., Lin, F., and Lafortune, S. (2012). On the computation of supremal sublanguages relevant to supervisory control. 45(29):175 – 180. WODES 2012.

[Moor et al., 2011] Moor, T., Schmidt, K., and Wittmann, T. (2011). Abstraction-based control for not necessarily closed behaviours. *Proceedings of the 18th IFAC World Congress*, pages 6988–6993.

[Paoli et al., 2011] Paoli, A., Sartini, M., and Lafortune, S. (2011). Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, 47(4):639–649.

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press.

[Pnueli and Rosner, 1989] Pnueli, A. and Rosner, R. (1989). On the synthesis of a reactive module. In *SIGPLAN-SIGACT*, pages 179–190, New York. ACM.

[Rabin, 1972] Rabin, M. O. (1972). *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Boston.

[Ramadge, 1989] Ramadge, P. J. (1989). Some tractable supervisory control problems for discrete-event systems modeled by büchi automata. *IEEE Transactions on Automatic Control*, 34:10–19.

[Ramadge and Wonham, 1987] 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.

[Ramadge and Wonham, 1989] Ramadge, P. J. and Wonham, W. M. (1989). Modular control of discrete event systems. *Maths. of Control, Signals & Systems*, 1:1:13–30.

[Safra, 1988] Safra, S. (1988). On the complexity of omega-automata. In *FOCS'88*, pages 319–327.

[Schmidt et al., 2008] Schmidt, K., Moor, T., and Perk, S. (2008). Nonblocking hierarchical control of decentralized discrete event systems. *IEEE Transactions on Automatic Control*, 53(10):2252–2265.

[Thistle, 1995] Thistle, J. G. (1995). On control of systems modelled as deterministic rabin automata. *Discrete Event Dynamic Systems*, 5(4):357–381.

[Thistle and Wonham, 1994a] Thistle, J. G. and Wonham, W. M. (1994a). Control of infinite behavior of finite automata. *SIAM J. Control and Optimization*, 32:1075–1097.

[Thistle and Wonham, 1994b] Thistle, J. G. and Wonham, W. M. (1994b). Supervision of infinite behavior of discrete event systems. *SIAM J. Control and Optimization*, 32:1098–1113.

[Thomas, 1990] Thomas, W. (1990). Automata on infinite objects. In *Handbook of Theoretical Computer Science (Vol. B)*, pages 133–191. MIT Press.

[Thomas, 1995] Thomas, W. (1995). On the synthesis of strategies in infinite games. In *STACS'95 Munich, Germany*, pages 1–13.

[Vardi and Wolper, 1986] Vardi, M. Y. and Wolper, P. (1986). Automata-theoretic techniques for modal logics of programs. *Journal of computer and system sciences*, 32:183–221.

[Wen et al., 2008] Wen, Q., Kumar, R., Huang, J., and Liu, H. (2008). A framework for fault-tolerant control for discrete event systems. *IEEE Transactions on Automatic Control*, 53:1839–1849.

[Willems, 1991] Willems, J. C. (1991). Paradigms and puzzles in the theory of dynamic systems. *IEEE Transactions on Automatic Control*, 36:258–294.

[Wong and Wonham, 1996] Wong, K. C. and Wonham, W. M. (1996). Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6(3):241–273.

[Yin and Lafortune, 2016] Yin, X. and Lafortune, S. (2016). Synthesis of maximally permissive supervisors for partially-observed discrete-event systems. *IEEE Trans. Automat. Contr.*, 61(5):1239–1254.

[Zhong and Wonham, 1990] Zhong, H. and Wonham, W. M. (1990). On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on Automatic Control*, 35:1125–1134.

[Zielonka, 1998] Zielonka, W. (1998). Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183.

A. Behavioural Characterisations of Problem Statements

The proofs for the behavioural characterization of Problem 1 and Problem 2 in Lemma 1 and Lemma 2, respectively, are very similar and rather straight forward. We provide them here in the interest of a complete and self-contained presentation.

Proof of Lemma 1. We prove both directions separately.

"⇒" Let $r : U^+ \to Y$ be a reactive module with associated behavior $\mathcal{L} \subseteq (UY)^\omega$. Consider the $*$-language

$$L_r := \{\, s \in (UY)^* \mid \forall\, t \in (U \cup Y)^*, y \in Y : \; sy \leq t \;\Rightarrow y = r(\mathrm{p}_U\, s) \,\}, \tag{46}$$

to observe $(\operatorname{pfx} \mathcal{L}) \cap (UY)^* \subseteq L_r$. By $\mathcal{L} \subseteq (UY)^\omega$, this implies $\operatorname{clo} \mathcal{L} = \lim \operatorname{pfx} \mathcal{L} \subseteq \lim L_r$. Moreover, $s \in L_r$ implies $(\operatorname{pfx} s) \cap (UY)^* \subseteq L_r$. In particular, we obtain $\lim L_r \subseteq \mathcal{L}$ and, hence, $\lim L_f = \mathcal{L}$.
(RM1) From the preliminary considerations, we observe $\operatorname{clo} \mathcal{L} \subseteq \lim L_r = \mathcal{L}$, and, hence, $\mathcal{L}$ is topologically closed.
(RM2) We first show that for any $s \in (\operatorname{pfx} \mathcal{L}) \cap (UY)^*$ and for any $u \in U$ it follows that $su \in \operatorname{pfx} \mathcal{L}$. Consider a sequence $(s_i)_{i \in \mathbb{N}}$ such that $s_i \in L_r$ for some $i \in \mathbb{N}$ and define $s_{i+1} := s_i u r(\mathrm{p}_U\, s_i u) \in L_r$. We initialise our construction with $s \in (\operatorname{pfx} \mathcal{L}) \cap (UY)^* \subseteq L_r$ to obtain $s_i \in L_r$ for all $i \in \mathbb{N}$. By construction, $(s_i)_{\mathbb{N}}$ is strictly monotone and, hence the limit amounts to a singleton $\alpha \in (UY)^\omega$, $\{\, \alpha \,\} = \lim\{\, s_i \mid i \in \mathbb{N} \,\} \subseteq \lim L_r = \mathcal{L}$. Observe that $su \leq s_2 < \alpha$, to conclude $su \in \operatorname{pfx} \mathcal{L}$. As u was arbitrary, we can now consider any $su' \in \operatorname{pfx} \mathcal{L}$ and pick an arbitrary $u'' \in U$ to see that $su'' \in \operatorname{pfx} \mathcal{L}$ follows from the above construction.
(RM3) Pick any $s \in \operatorname{pfx} \mathcal{L}$ and any $y', y'' \in Y$ such that $sy', sy' \in \operatorname{pfx} \mathcal{L}$. We need to show that $y' = y''$. As $sy', sy' \in \operatorname{pfx} \mathcal{L}$, we have $sy', sy' \in L_r$ and, hence, $y' = r(\mathrm{p}_U\, s) = y''$.
(non-emptiness) Observe the $\epsilon \in L_r$. Hence, we can pick an arbitrary $u \in U$ and do the same construction as in (RM2), now staring with $s_1 = \epsilon$, to obtain $\alpha \in \mathcal{L}$.

"⇐" Given a non-empty language $\mathcal{L} \subseteq (UY)^\omega$ that complies to (RM1) to (RM3), construct a reactive module $r : U^+ \to Y$ with associated behavior $\mathcal{L}$. We prove this claim in three steps (A)-(C).
(A) Show that for any $v \in U^+$ there uniquely exist $s \in (UY)^*U$ and $y \in Y$ such that

$$\mathrm{p}_U\, s = v \quad \text{and} \quad sy \in \operatorname{pfx} \mathcal{L}\,. \tag{47}$$

We establish the claim by induction over the length of v.
– For $v \in U$, non-emptiness of $\mathcal{L}$ and (RM2) implies $v \in \operatorname{pfx} \mathcal{L}$. In particular, we can choose $y \in Y$ such that $vy \in \operatorname{pfx} \mathcal{L}$. By (RM3), the latter choice of y is unique and, with $s := v$ this establishes the claim for input strings v of length 1.
– Now assume that the claim holds for some $v \in U^+$ and consider vu for an arbitrary $u \in U$. Let $s \in (UY)^*U$ and $y \in Y$ denote the unique choice to satisfy Eq. (47) for v. By (RM2), we obtain $syu \in \operatorname{pfx} \mathcal{L}$, and, hence, there exists $y' \in Y$ such that $syuy' \in \operatorname{pfx} \mathcal{L}$. With $s' = syu$, we observe $\mathrm{p}_U\, s' = vu$ and $s'y' \in \operatorname{pfx} \mathcal{L}$. Now consider any $s'' \in (U \cup Y)^*$ and $y'' \in Y'$ with $\mathrm{p}_U\, s'' = vu$ and $s''y'' \in \operatorname{pfx} \mathcal{L}$. Since the length of vu is at least 2, we can decompose by $s'' = tu'''y'''u$ with $u''' \in U$ and $y''' \in Y$. By the induction hypothesis, we obtain $t'u''' = s$ and $y''' = y$. Hence, $s'' = syu = s'$. Together with (RM3), this also implies $y'' = y'$.
(B) Given that the claim of step (A) is satisfied we can define functions $f : U^+ \to (U \cup Y)^*$ and $r : U^+ \to Y$ with domain U^+ such that Eq. (47) is true for $s \in (UY)^*U$

and $y \in Y$ if and only if $s = f(v)$ and $y = r(v)$. Clearly, r is a reactive module. (C) Show that the associated behavior of r constructed in step (B) matches $\mathcal{L}$: Pick any $\alpha \in \mathcal{L}$ and consider any prefix $sy < \alpha$ with $s \in (UY)^*U$ and $y \in Y$. Denote $v = \mathrm{p}_U\, s$. By $sy \in \operatorname{pfx}\mathcal{L}$ and the claim in (A) this implies $s = f(v)$ and $y = r(v)$. Since this holds true for any prefix, α complies with the behavior associated with r. For the converse inclusion, let $\alpha \in (UY)^\omega$ belong to the behavior associated with r. By $s_i y_i < \alpha$, $s_i \in (UY)^{(i-1)}U$, $y_i \in Y$, $i \in \mathbb{N}$, we construct a strictly monotone sequence of prefixes of α. Observe via (RM2) that $s_1 \in \operatorname{pfx}\mathcal{L}$. Now assume $s_i \in \operatorname{pfx}\mathcal{L}$ for some $i \in \mathbb{N}$ and consider $v := \mathrm{p}_U\, s_i$. By the claim in (A), there uniquely exists $s \in (UY)^*U$ and $y \in Y$ such that Eq. (47) holds, where we have $y = r(v)$ and $s = f(v)$. Uniqueness then implies $s_i = s$ and, referring to the definition of the associated behavior, we also have $y_i = r(v)$. We conclude $s_i y_i \in \operatorname{pfx}\mathcal{L}$, and hence, by (RM2), $s_{i+1} \in \operatorname{pfx}\mathcal{L}$. Thus, infinitely many prefixes $(s_i)_{i\in\mathbb{N}}$ of α are within $\operatorname{pfx}\mathcal{L}$. Topological closedness (RM1) then implies $\alpha \in \mathcal{L}$.

□

Proof of Lemma 2. We prove both directions separately.

"⇒" Let $f : \Sigma^* \to \Gamma$ be a supervisor with associated behavior $\mathcal{L}$. Consider the $*$-language

$$L_f := \{\, s \in \Sigma^* \,|\, \forall\, t \in \Sigma^*, \sigma \in \Sigma :\ s\sigma \leq t\ \Rightarrow \sigma \in f(s) \,\}, \tag{48}$$

to observe $\operatorname{pfx}\mathcal{L} \subseteq L_f$ and, hence, $\operatorname{clo}\mathcal{L} = \lim \operatorname{pfx}\mathcal{L} \subseteq \lim L_f$. Moreover, $s \in L_f$ implies $\operatorname{pfx} s \subseteq L_f$, i.e., L_f is prefix-closed and we conclude that $\lim L_f \subseteq \mathcal{L}$. This amounts to $\lim L_f = \mathcal{L}$.
(SC1) As the limit of a prefix-closed $*$-language, $\mathcal{L}$ is topologically closed.
(SC2) Pick any $s \in \operatorname{pfx}\mathcal{L}$ and any $\sigma \in \Sigma_{uc}$ and show that $s\sigma \in \operatorname{pfx}\mathcal{L}$. Consider a sequence $(s_i)_{i\in\mathbb{N}}$ and assume that $s_i \in L_f$ for some $i \in \mathbb{N}$. As $\emptyset \neq \Sigma_{\mathrm{uc}} \subseteq f(s_i)$, the property $s_i \in L_f$ implies that $s_{i+1} := s_i\sigma \in L_f$. Furthermore, as $s \in \operatorname{pfx}\mathcal{L} \subseteq L_f$ we can choose $s_1 := s$ to initialise the sequence. By construction, $(s_i)_{\mathbb{N}}$ is strictly monotone and, hence, the limit amounts to a singleton $\alpha \in \Sigma^\omega$ with $\{\,\alpha\,\} = \lim\{\, s_i \,|\, i \in \mathbb{N}\,\} \subseteq \lim L_f = \mathcal{L}$. Observe that $s\sigma = s_2 < \alpha$, to conclude $s\sigma \in \operatorname{pfx}\mathcal{L}$.
(non-emptiness) Observe that $\epsilon \in L_r$. Since Σ_{uc} we can pick an arbitrary $\sigma \in \Sigma_{\mathrm{uc}}$ and conduct the same construction as in (SC2), now starting with $s_1 = \epsilon$, to obtain $\alpha \in \mathcal{L}$.

"⇐" Given a non-empty language $\mathcal{L} \subseteq \Sigma^\omega$ s.t. (SC1) and (SC2) hold, we construct the canditate supervisor $f : \Sigma^* \to \Gamma$ defined by

$$f'(s) := \{\, \sigma \in \Sigma \,|\, s\sigma \in \operatorname{pfx}\mathcal{L} \,\} \cup \Sigma_{\mathrm{uc}} \tag{49}$$

and show that its associated behaviour $\mathcal{L}'$ defined via Eq. (15) coincides with $\mathcal{L}$.
$\mathcal{L} \subseteq \mathcal{L}'$: Pick any $\alpha \in \mathcal{L}$ and consider an arbitrary prefix $s\sigma < \alpha$ with $s \in \Sigma^*$, $\sigma \in \Sigma$. In particular, we have $s\sigma \in \operatorname{pfx}\mathcal{L}$ and, hence, $\sigma \in f'(s)$. By the arbitrary choice of the prefix, we conclude that $\alpha \subseteq \mathcal{L}'$ from Eq. (15).
$\mathcal{L}' \subseteq \mathcal{L}$: Pick any $\alpha \in \mathcal{L}'$. We fist show that $\operatorname{pfx}\alpha \subseteq \operatorname{pfx}\mathcal{L}$ by induction over the length of the respective prefix. Clearly, the claim is true for the prefix of length 0, i.e., we have $\epsilon \in \operatorname{pfx}\mathcal{L}$ by non-emptiness of $\mathcal{L}$. For the induction step, consider $s \in \operatorname{pfx}\mathcal{L}$ and $\sigma \in \Sigma$ with $s\sigma < \alpha$. Since $\alpha \in \mathcal{L}'$ we have that $\sigma \in f(s)$ (from Eq. (15)). If σ is in the left union component of Eq. (49), we directly obtain $s\sigma \in \operatorname{pfx}\mathcal{L}$. If σ is in the right union component Σ_{uc}, we also obtain $s\sigma \in \operatorname{pfx}\mathcal{L}$ from the proof of (SC1). Hence, $\operatorname{pfx}\alpha \subseteq \operatorname{pfx}\mathcal{L}$ and we finally obtain $\alpha \in \mathcal{L}$ by topological closeness (SC1). □

B. Fixed-Point Characterisation of the Controllability Prefix

Although the synthesis algorithm for supervisory controllers via the controllability prefix as used in this study is a specialization of the one provided by [Thistle and Wonham, 1994a], we believe that it is instructive to provide a self-contained and independent proof[12] of Theorem 1. We discuss both directions of Eq. (26) separately.

B.1. "⇐:" From $\mathsf{Win}(M)$ to $\mathrm{cfx}_{\mathcal{A}}\,\mathcal{G}$

Consider the sets Z^∞, Y^i, X^i, W^i_j for $0 \leq i \leq k$ and $0 \leq j \leq l$ as constructed in Section 3.4 and assume that we redo the calculation of the fixed-point in (24) after Z^∞ has been computed in the previous iteration. Under the assumption that the iteration over X has already converged, i.e., $X^i = Y^i$, we consider the last iteration of the inner-most fixed-point of (24), which is evaluated by iteratively computing

$$W^i_j = \mathsf{Pre}((W^i_{j-1} \setminus F_{\mathcal{A}}) \cup Y^{i-1} \cup (F_{\mathcal{G}} \cap Z^\infty), Y^i \setminus F_{\mathcal{A}}) \tag{50}$$

initialized with $Y^0 = \emptyset$ and W^i_j. This equation can be interpreted as follows. Given a state $q \in Z^\infty$ with $\mathsf{rank}(q) = (i,j) > (0,0)$ we know that $q \in W^i_j$ from (27). Now (50) tells us (via the definition of Pre in (25)) that there exists a transition (q, σ, q') s.t. either
(a) $q' \in W^i_{j-1} \setminus F_{\mathcal{A}}$ (resulting in $\mathsf{rank}(q') \leq (i, j-1)$ and $q' \in Z^\infty$), or
(b) $q' \in Y^{i-1}$ (resulting in $\mathsf{rank}(q') \leq (i-1, \cdot)$ and $q' \in Z^\infty$), or
(c) $q' \in (F_{\mathcal{G}} \cap Z^\infty)$ (resulting in $\mathsf{rank}(q') = (0,0)$ and $q' \in Z^\infty$).
Furthermore, we know for any uncontrollable transition (q, σ', q') with $\sigma' \in \Sigma_{uc}$ that either cases (a)-(c) is true or we have
(d) $q' \in Y^i \setminus F_{\mathcal{A}}$ (resulting in $\mathsf{rank}(q') \leq (i, \cdot)$ and $q' \in Z^\infty$).
This observation is the essential ingredient in all proofs of lemmas utilized in this section.

Now let f be a supervisor constructed via g in (28) and $\mathcal{L}$ its associated behaviour defined via (15). We start by proving the two properties of f in Proposition 2, namely, non-blockingness and soundness, separately. After that, we proceed with the actual proof of the "⇐" part of (26).

Non-conflictingness Given the constructed supervisor f and a string $s \in \Sigma^*$ s.t. $q = \delta(q_0, s) \in Z^\infty$, let f restrict the behaviour of the plant after q was reached. It is easy to see that the supervisor f constructed via g in (28) only enables uncontrollable transitions (which always keep the system in Z^∞ due to cases (a)-(d)) or (if they exist) controllable transitions resulting in a state q' for which a ranking is defined, which is only true for $q' \in Z^\infty$. Hence, it is easy to see that the closed loop formed by the plant $\mathcal{A}$ and f always keeps the system in Z^∞. This is formalized by the following lemma.

Lemma 3. Let $s \in \Sigma^*$ s.t. $\delta(q_0, s) \in Z^\infty$. Then it holds for all $\alpha \in \Sigma^*$ that

$$s\alpha \in (\mathrm{pfx}\,\mathcal{L}) \cap (\mathrm{pfx}\,\mathcal{A}) \Rightarrow \delta(q_0, s\alpha) \in Z^\infty. \tag{51}$$

Proof. Let ρ be the corresponding run of α on M with $\rho(0) = \delta(q_0, s) \in Z^\infty$. Hence, there exists $i, j \geq 1$ s.t. either $\mathsf{rank}(\rho(0)) = (i,j) > (0,0)$, i.e., $\rho(0) \in W^i_j \subseteq Y^i \subseteq Z^\infty$

[12] This proof is inspired by a discussion with Nir Piterman during a Dagstuhl Seminar in May 2017.

or $\mathsf{rank}(\rho(0)) = (0,0)$, i.e., $\rho(0) \in Z^\infty \cap F_\mathcal{G} \subseteq Z^\infty$. Furthermore, we know that $\alpha(0) \in f(s) = g(\rho(0))$ with $\rho(1) = \delta(\rho(0), \alpha(0))$. Given this, the claim follows by induction: assuming that $\mathsf{rank}(\rho(l)) = (i,j)$ and hence $\rho(l) \in Z^\infty$ we know from the above reasoning about f that $\alpha(l) \in f(s\alpha|_{[0,l-1]}) = g(\rho(l))$ and $\rho(l+1) = \delta(\rho(l), \alpha(l))$ implies $\rho(l+1) \in Z^\infty$. □

Intuitively, the above interpretation of (50) shows that when using f to control $\mathcal{A}$ there always exists the posibility to decrease the rank with the next transition (cases (a)-(b)). As l and k are finite, this implies that we eventually reach a state q with rank $(0,0)$, implying $q \in F_\mathcal{G}$. In this case any transition that keeps the system in Z^∞ (which always exists) is applied and the ranking is reset, allowing for the same reasoning as before. This shows that there always exists a run ρ on M which is allowed by f and visits $F_\mathcal{G}$ infinitely often. This is formally stated in the next lemma.

Lemma 4. For all $s \in \Sigma^*$ holds

$$\delta(q_0, s) \in Z^\infty \Rightarrow \exists \beta \in \Sigma^\omega \,.\, \begin{pmatrix} s\beta \in \mathcal{L} \cap (\operatorname{clo} \mathcal{A}) \\ \wedge s\beta \in \mathcal{G} \end{pmatrix}. \tag{52}$$

Proof. Let $s' := s$ and $q := (q_0, s') \in Z^\infty$ implying the existence of $i, j > 0$ s.t. $q \in W^i_j$. Hence, we know that we can pick a particular $\sigma \in \Sigma$ which is enabled in q s.t. the post state $q' = \delta(q, \sigma)$ fulfills at least one of the cases (a)-(c). If $\sigma \in \Sigma_{uc}$ it is easy to see that $\sigma \in g(q)$ and hence $s\sigma \in (\operatorname{pfx} \mathcal{L}) \cap (\operatorname{pfx} \mathcal{A})$. Now assume $\sigma \in \Sigma_c$ and recall that $\mathsf{rank}(q) = (i,j)$. If $(i,j) > (0,0)$ we see that cases (a),(b) and (c) correspond to $\mathsf{rank}(q') < \mathsf{rank}(q)$ and hence $\sigma \in g(q)$. If $(i,j) = (0,0)$ we see that for all cases $q' \in Z^\infty$ and hence $\sigma \in g(q)$.

By resetting s' to $s\sigma$ we can redo the above transition construction infinitely often to obtain a state sequence ρ corresponding to the word $s\beta$ (with $\beta(l)$ being the σ selected in the lth iteration of the above reasoning and $\rho(l+1) = \delta(\rho(l), \beta(l))$). It is easy to see that for all $l \in \mathbb{N}$ holds $\rho(l) \in Z^\infty$ by construction and therefore there exists i and j s.t. $\rho(l) \in W^i_j$ and furthermore $s\beta \in \lim((\operatorname{pfx} \mathcal{L}) \cap (\operatorname{pfx} \mathcal{A})) = \mathcal{L} \cap (\operatorname{clo} \mathcal{A})$. It remains to show that $s\beta \in \mathcal{G}$.

First, observe that whenever $\mathsf{rank}(\rho(l)) > (0,0)$ the rank is decreased by any transition corresponding to cases (a),(b) and (c). As by definition of Pre such a transition always exists we eventually reach an l' s.t. $\mathsf{rank}(\rho(l')) = (0,0)$ implying $\rho(l') \in F_\mathcal{G} \cap Z^\infty$. In this case g allows for any transition that keeps ρ in Z^∞ and therefore resets the ranking of $\rho(l'+1)$ to some (i,j). Applying this reasoning infinitely often yields infinitely many visits to $F_\mathcal{G}$ and therefore implies $s\beta \in \mathcal{G}$. □

As an immediate consequence of Lemma 3 and Lemma 4 we can now show that the constructed supervisor is non-blocking, i.e., its induced behavior and the plant behavior are non-conflicting. Interestingly, this is only true for the supervisor f constructed via g in (28) if $\mathcal{G} \subseteq \mathcal{A}$.

Lemma 5. If $\mathcal{G} \subseteq \mathcal{A}$, the languages $\mathcal{L}$ and $\mathcal{A}$ are non-conflicting, i.e.,

$$\operatorname{pfx}(\mathcal{L} \cap \mathcal{A}) = (\operatorname{pfx} \mathcal{L}) \cap (\operatorname{pfx} \mathcal{A}). \tag{53}$$

Proof. Observe that "$\subseteq$" in (53) always holds. We therefore only prove the reverse direction. Pick $s\alpha \in (\operatorname{pfx} \mathcal{L}) \cap (\operatorname{pfx} \mathcal{A})$. As f in (28) is only defined for states in Z^∞ we know that this implies $\delta(q_0, s) \in Z^\infty$. With this it follows from Lemma 3 that

$\delta(q_0, s\alpha) \in Z^\infty$. Then we can use Lemma 4 to pick $\beta \in \Sigma^\omega$ s.t. $s\alpha\beta \in \mathcal{L} \cap \text{clo}(\mathcal{A})$ and $s\alpha\beta \in \mathcal{G}$. If $\mathcal{G} \subseteq \mathcal{A}$ we therefore have $s\alpha\beta \in \mathcal{A}$ and hence $s\alpha\beta \in \mathcal{L} \cap \mathcal{A}$. With this we immediately have that $s\alpha \in \text{pfx}(\mathcal{L} \cap \mathcal{A})$. □

Soundness of the Induced Closed Loop Behaviour While Lemma 4 shows that there exists a run ρ visiting $F_\mathcal{G}$ infinitely often, it should be noticed that due to the use of uncontrollable events, such a run might not be "enforceable" by the supervisor. However, using the property of the plant which is assumed to only generate runs corresponding to words in $\mathcal{A}$, it follows that for all such runs allowed by f, $F_\mathcal{G}$ is visited infinitely often.

Lemma 6. Let $s \in \Sigma^*$ s.t. $\delta(q_0, s) \in Z^\infty$. Then

$$s\alpha \in \mathcal{L} \cap \mathcal{A} \Rightarrow s\alpha \in \mathcal{G}. \tag{54}$$

If $q_0 \in Z^\infty$ this implies $\emptyset \neq \mathcal{L} \cap \mathcal{A} \subseteq \mathcal{G}$.

Proof. Pick $s\alpha \in \mathcal{L} \cap \mathcal{A}$ and let ρ be its corresponding run on M, i.e. $inf(\rho) \cap F_\mathcal{A} \neq \emptyset$. We need to show that for any $l \in \mathbb{N}$ s.t. $\rho(l) \in F_\mathcal{A}$ there exists a $j \geq l$ s.t. $\rho(j) \in F_\mathcal{G}$.

As $\rho(l) \in Z^\infty$ (from Lemma 3) there exist i, j s.t. $\mathsf{rank}(\rho(l)) = (i, j)$. Now recall that any $\sigma \in g(\rho(l))$ will result in a post state $q' = \rho(l+1)$ covered by one of the cases (a)-(d). As $inf(\rho) \cap F_\mathcal{A} \neq \emptyset$ we know that there exists a $l' \geq l$ s.t. $\rho(l') \in F_\mathcal{A}$. Now assume that there exists no $l < \tilde{l} \leq l'$ s.t. $\rho(\tilde{l}) \in F_\mathcal{G}$ (as otherwise $\mathsf{rank}(\rho(\tilde{l})) = (0, 0)$ and hence the ranking would have been reset). Then we know that $\mathsf{rank}(\rho(l')) < \mathsf{rank}(\rho(l))$ as $\rho(l') \in F_\mathcal{A}$ is only possible if $\rho(l')$ corresponds to case (b) or (c). If it corresponds to case (b) we still have $\mathsf{rank}(\rho(l')) > (0, 0)$ and can repeat the above reasoning. As we can only decrease the rank finately many times we see that we eventually obtain $\mathsf{rank}(\rho(l'')) = (0, 0)$ for some $l'' > l'$ and hence $\rho(l'') \in F_\mathcal{G}$. As this resets the ranking in ρ, we can repeat the above reasoning infinitely often and obtain $s\alpha \in \mathcal{G}$. □

Proving the "⇐" part of (26) Using Lemma 5 and Lemma 6 we can finally prove the "⇐" part of (26). For this purpose let $s \in \Sigma^*$ be s.t. $q = \delta(q_0, s) \in Z^\infty$ and construct $\mathcal{V}_s$ s.t.

$$\mathcal{V}_s := (s\Sigma^\omega) \cap \mathcal{L} \cap \mathcal{A}. \tag{55}$$

We now proceed by proving all required properties for $\mathcal{V}_s$ from Definition 2 implying that $s \in \text{cfx}_\mathcal{A}\mathcal{G}$:

• Show $\mathcal{V}_s \subseteq \mathcal{G} \cap (s\Sigma^\omega)$: Observe, that it follows directly from (55) and Lemma 6 that $\mathcal{V}_s \subseteq \mathcal{G}$. Furthermore, it follows from Lemma 4 that $s \in \text{pfx}(\mathcal{L} \cap \mathcal{A})$ and therefore $s \in \text{pfx}\,\mathcal{V}_s$ by the specific structure of (55). Combining both observations implies $\mathcal{V}_s \subseteq \mathcal{G} \cap (s\Sigma^\omega)$.

• Show $\mathcal{V}_s = (\text{clo}\,\mathcal{V}_s) \cap \mathcal{A}$: Relative closedness w.r.t. $\mathcal{A}$ is implied by $\mathcal{V}_s$ to be the intersection of a closed set with $\mathcal{A}$.

• Show $((\text{pfx}\,\mathcal{V}_s)\Sigma_{uc}) \cap ((\text{pfx}\,\mathcal{A}) \cap (s\Sigma^*)) \subseteq \text{pfx}\,\mathcal{V}_s$: Pick $s\alpha\sigma \in ((\text{pfx}\,\mathcal{V}_s)\Sigma_{uc}) \cap (\text{pfx}\,\mathcal{A}) \cap (s\Sigma^*)$. This implies $s\alpha\sigma \in (\text{pfx}\,\mathcal{A})$ and $s\alpha \in \text{pfx}\,\mathcal{L}$. Then it follows from the construction of f via g in (28) that for any $\sigma \in \Sigma_{uc}$ we have $s\alpha\sigma \in \text{pfx}\,\mathcal{L}$ and therefore $s\alpha\sigma \in \text{pfx}\,\mathcal{L}$. As $s\alpha\sigma \in \text{pfx}\,\mathcal{A}$ this implies $s\alpha\sigma \in (\text{pfx}\,\mathcal{L}) \cap (\text{pfx}\,\mathcal{A})$. Then it follows from Lemma 5 that $s\alpha\sigma \in \text{pfx}\,\mathcal{V}_s$, which proves the statement.

B.2. "⇒:" From $\mathrm{cfx}_{\mathcal{A}}\,\mathcal{G}$ to $\mathsf{Win}(M)$

Down the road, we will prove this direction in (26) by contradiction. For this, we need to construct the complement of Z^∞ by negating the fixed-point in (24).

Negating the fixed-point in (24) Consider the existential and controllable Pre-operators

$$\begin{aligned}\mathsf{Pre}_e(T) &:= \{q \in Q \mid \exists \sigma \in \Sigma \,.\, \delta(q,\sigma) \in T\}\\ \mathsf{Pre}_c(T) &:= \{q \in Q \mid \forall \sigma \in \Sigma_{uc} \cap \mathsf{Enab}(q) \,.\, \delta(q,\sigma) \in T\}\end{aligned}$$

and their negated versions

$$\begin{aligned}\neg\mathsf{Pre}_e(T) = \mathsf{Pre}_u(\neg T) &:= \{q \in Q \mid \forall \sigma \in \Sigma \cap \mathsf{Enab}(q) \,.\, \delta(q,\sigma) \in \neg T\}\\ \neg\mathsf{Pre}_c(T) = \mathsf{Pre}_{uc}(\neg T) &:= \{q \in Q \mid \exists \sigma \in \Sigma_{uc} \,.\, \delta(q,\sigma) \in \neg T\}\end{aligned}$$

where Pre_u and Pre_{uc} denote the universal and un-controllable Pre-operators, respectively. Then it is easy to see that the inverse dynamics operator $\mathsf{Pre}(T,\, D)$ defined in (25) can be decomposed in an existential and a controllable part and we have

$$\mathsf{Pre}(T,\, D) = \mathsf{Pre}_e(T) \cap \mathsf{Pre}_c(T \cup D).$$

With this the fixed-point in (24) becomes

$$\begin{aligned}\nu Z \,.\, \mu Y \,.\, \nu X \,.\, \mu W \,.\, &\mathsf{Pre}_e((W \setminus F_{\mathcal{A}}) \cup Y \cup (F_{\mathcal{G}} \cap Z))\\ &\cap \mathsf{Pre}_c((X \setminus F_{\mathcal{A}}) \cup Y \cup (F_{\mathcal{G}} \cap Z))\end{aligned} \tag{56}$$

as we always have $W \subseteq X$ during the fixed-point iterations.

We now use the negation rule of the μ-calculus, i.e., $\neg(\mu X \,.\, F(X)) = \nu \overline{X} \,.\, \overline{F}(\overline{X})$ where over-lined sets and functions denote there negation, to negate (56). This results in the fixed-point

$$\begin{aligned}&\mu\overline{Z} \,.\, \nu\overline{Y} \,.\, \mu\overline{X} \,.\, \nu\overline{W} \,.\, \mathsf{Pre}_u(\neg((W \cap \overline{F}_{\mathcal{A}}) \cup Y \cup (F_{\mathcal{G}} \cap Z)))\\ &\qquad\qquad \cup \mathsf{Pre}_{uc}(\neg((X \cap \overline{F}_{\mathcal{A}}) \cup Y \cup (F_{\mathcal{G}} \cap Z)))\\ &= \mu\overline{Z} \,.\, \nu\overline{Y} \,.\, \mu\overline{X} \,.\, \nu\overline{W} \,.\, \mathsf{Pre}_u((\overline{W} \cup F_{\mathcal{A}}) \cap \overline{Y} \cap (\overline{F}_{\mathcal{G}} \cup \overline{Z})))\\ &\qquad\qquad \cup \mathsf{Pre}_{uc}((\overline{X} \cup F_{\mathcal{A}}) \cap \overline{Y} \cap (\overline{F}_{\mathcal{G}} \cup \overline{Z}))\\ &= \mu\overline{Z} \,.\, \nu\overline{Y} \,.\, \mu\,\overline{X} \,.\, \nu\,\overline{W} \,.\, \mathsf{Pre}_u(\overline{Z} \cup (\overline{W} \cap \overline{F}_{\mathcal{G}}) \cup (\overline{Y} \cap F_{\mathcal{A}} \cap \overline{F}_{\mathcal{G}})))\\ &\qquad\qquad \cup \mathsf{Pre}_{uc}(\overline{Z} \cup (\overline{X} \cap \overline{F}_{\mathcal{G}}) \cup (\overline{Y} \cap F_{\mathcal{A}} \cap \overline{F}_{\mathcal{G}}))\end{aligned} \tag{57}$$

where the last line of the derivation follows from $\overline{Z} \subseteq \overline{X} \subseteq \overline{W} \subseteq \overline{Y}$ given by the properties of the μ-calculus.

Let $\overline{Z}^\infty$ denote the fixed-point computed by (57). Then $\overline{Z}^i$ denotes the set obtained in the ith iteration over $\overline{Z}$ and we denote by $\overline{Y}^i = \overline{Z}^i$ the corresponding fixed-point of the iteration over $\overline{Y}$. Furthermore, we denote by $\overline{X}^i_j$ the set obtained in the jth iteration over $\overline{X}$ performed while computing $\overline{Y}^i$. Then it follows from the properties of the fixed point that after the ith iteration over $\overline{Z}$ has terminated, we have $\overline{Z}^i = \bigcup_j \overline{X}^i_j$ (in particular $\overline{Z}^k = \bigcup_j \overline{X}^k_j$ for $\overline{Z}^\infty = \overline{Z}^k$) and the algorithm is initialized with $\overline{Z}^0 = \emptyset$ and $\overline{X}^0_0 = \emptyset$. With this we see that after the termination of

the inner fixed-point over $\overline{W}$ the following holds for all states in $\overline{X}^i_j$:

$$\begin{aligned}\overline{X}^i_j = &\mathsf{Pre}_u(\overline{Z}^{i-1} \cup (\overline{X}^i_j \cap \overline{F}_\mathcal{G}) \cup (\overline{Z}^i \cap F_\mathcal{A} \cap \overline{F}_\mathcal{G}))) \\ &\cup \mathsf{Pre}_{uc}(\overline{Z}^{i-1} \cup (\overline{X}^i_{j-1} \cap \overline{F}_\mathcal{G}) \cup (\overline{Z}^i \cap F_\mathcal{A} \cap \overline{F}_\mathcal{G})).\end{aligned} \tag{58}$$

Now we can interpret (58) similar to (50). Given a state $q \in \overline{Z}^\infty = \overline{Z}^k$ we know that there exists a j s.t. $q \in X^k_j$. Now (58) tells us (via the definition of Pre_u and Pre_{uc}) that one of the following two cases is true.

(1) *All* events enabled in q lead to a next state $q' = \delta(q, \sigma)$ s.t. either
  (1a) $q' \in \overline{Z}^{k-1} \subseteq \overline{Z}^\infty$,
  (1b) $q' \in \overline{X}^k_j \cap \overline{F}_\mathcal{G} \subseteq \overline{Z}^\infty$, or
  (1b) $q' \in \overline{Z}^k \cap F_\mathcal{A} \cap \overline{F}_\mathcal{G} \subseteq \overline{Z}^\infty$, or
(2) there *exists* an uncontrollable event $\sigma \in \Sigma_{uc}$ enabled in q which leads to the next state $q' = \delta(q, \sigma)$ s.t. either
  (2a) $q' \in \overline{Z}^{k-1} \subseteq \overline{Z}^\infty$,
  (2b) $q' \in \overline{X}^k_{j-1} \cap \overline{F}_\mathcal{G} \subseteq \overline{Z}^\infty$, or
  (2b) $q' \in \overline{Z}^k \cap F_\mathcal{A} \cap \overline{F}_\mathcal{G} \subseteq \overline{Z}^\infty$.

This observation is again essential for all proofs in this section.

Invariance of $\overline{Z}^\infty$ Consider a string $t \in \Sigma^*$ and any behaviour $\tilde{\mathcal{V}}_t \subseteq \mathcal{A} \cap (t\Sigma^\omega)$ which fulfills properties (i) and (ii) in Definition 2. Then we can use (21) to construct a map

$$\tilde{f}(s) := \{\, \sigma \in \Sigma \mid ts\sigma \in \operatorname{pfx} \tilde{\mathcal{V}}_t \,\} \cup \Sigma_{\text{uc}} \quad \text{for all } s \in \Sigma^* \tag{59}$$

with associated behaviour $\tilde{\mathcal{L}}$ defined via (15). We now show that this map renders $\overline{Z}^\infty$ invariant, i.e., we show that given a string $s \in \Sigma^*$ s.t. $\delta(q_0, s) \in \overline{Z}^\infty$ (i.e., $\delta(q_0, s) \notin Z^\infty$), there exists a path in the behaviour of M restricted by $\tilde{f}$ which will keep the system in $\overline{Z}^\infty$ and only visits $F_\mathcal{G}$ finitely often.

Lemma 7. Let $s \in \Sigma^*$ s.t. $\delta(q_0, s) \in \overline{Z}^\infty$. Then

$$\exists s\beta \in \tilde{\mathcal{L}} \cap \mathcal{A} \,.\, \begin{pmatrix} \forall l \in \mathbb{N} \,.\, \delta(q_0, s\beta|_{[0,l]}) \in \overline{Z}^\infty \\ \wedge s\beta \notin \mathcal{G} \end{pmatrix}. \tag{60}$$

Proof. Define $s' := s$ and $q := (q_0, s') \in \overline{Z}^\infty = \overline{Z}^k$, which implies the existence of j s.t. $q \in X^k_j$, i.e. (58) holds. If the first line of (58) holds, we can pick any σ enabled in q (in particular one ensuring $s\sigma \in \operatorname{pfx}(\tilde{\mathcal{L}} \cap \mathcal{A})$) s.t. the post state $q' = \delta(q, \sigma)$ fulfills one of the cases (1a)-(1c). If the second line of (58) holds, we know that there exists $\sigma \in \Sigma_{uc}$ s.t. the post state $q' = \delta(q, \sigma)$ fulfills one of the cases (2a)-(2c). To establish that for any such $\sigma \in \Sigma_{uc}$ we have $s\sigma \in \operatorname{pfx}(\tilde{\mathcal{L}} \cap \mathcal{A})$, recall that (SC2) holds for $\tilde{\mathcal{L}}$ and therefore $s\sigma \in \operatorname{pfx} \tilde{\mathcal{L}}$. As $s\sigma \in \operatorname{pfx} \mathcal{A}$ by the definition of M, it follows that $s\sigma \in \operatorname{pfx}(\tilde{\mathcal{L}} \cap \mathcal{A})$.

By resetting s' to $s\sigma$ we can redo the above transition construction infinitely often to obtain a state sequence ρ corresponding to the word $s\beta$ (with $\beta(l)$ being the σ selected in the lth iteration of the above reasoning and $\rho(l+1) = \delta(\rho(l), \beta(l))$). It is easy to see that for all $l \in \mathbb{N}$ holds $\rho(l) \in \overline{Z}^\infty$ by construction and therefore there exists i and j s.t. $\rho(l) \in \overline{X}^i_j$ and furthermore $s\beta \in \operatorname{clo}(\tilde{\mathcal{L}} \cap \mathcal{A})$. It remains to show that $s\beta \notin \mathcal{G}$ and $s\beta \in \tilde{\mathcal{L}} \cap \mathcal{A}$.

First, observe that whenever (1a) or (2a) is true in iteration l the i rank of $\rho(l)$ is decreased by one. Similarily, whenever (2b) is true, the j rank of $\rho(l)$ is decreased. As the iteration bound over $\overline{Z}$ and $\overline{X}$ is finite and $\overline{Z}^0 = \emptyset$ we can only go through cases (1a), (2a) and (2b) finitely often. This immediately implies $s\beta \notin \mathcal{G}$. Furthermore, as $s\beta \in \mathrm{clo}(\tilde{\mathcal{L}} \cap \mathcal{A})$ we know that for any $\rho(l)$ there exists a path to $F_{\mathcal{A}}$. As the second line of (58) eventually reduces to case (2c), implying that $F_{\mathcal{A}}$ is visited whenever an enabled uncontrollerable event is applied in $\rho(l)$ the universal pre-operator in the first line of (58) ensures that we can always eventually select a transition visiting $F_{\mathcal{A}}$ (as this possibility is retained given that $s\beta \in \mathrm{clo}(\tilde{\mathcal{L}} \cap \mathcal{A})$). This shows that we can construct $s\beta$ s.t. $s\beta \in \tilde{\mathcal{L}} \cap \mathcal{A}$, what proves the statement. □

Proving the "⇒" part of (26) We now proceed to the actual proof of the "⇒" part of (26) using the fact that $t \in \mathrm{cfx}_{\mathcal{A}} \mathcal{G}$ and constructing a supervisory candidate via (59) for the behaviour $\tilde{\mathcal{V}}_t$ associated with t via Definition 2.

Lemma 8. Let $t \in \mathrm{cfx}_{\mathcal{A}} \mathcal{G}$ and $\tilde{\mathcal{V}}_t \subseteq \mathcal{G} \cap (t\Sigma^\omega)$ s.t. $\tilde{\mathcal{V}}_t$ fulfills properties (i) and (ii) in Definition 2. Furthermore, let $\tilde{f}$ be a supervisor defined for $\tilde{\mathcal{V}}_t$ via (59) and let $\tilde{\mathcal{L}}$ denote its associated behaviour. Then it holds that

$$\tilde{\mathcal{V}}_t = \tilde{\mathcal{L}} \cap \mathcal{A}. \tag{61}$$

Proof. NWe show both directions of (61) separately:
"⊆" Pick $\alpha \in \tilde{\mathcal{V}}_t$. Then it follows from condition (i) in Definition 2 that $\alpha \in \mathcal{A}$ and $\alpha \in \mathrm{clo}(\tilde{\mathcal{V}}_t)$. As $\mathrm{clo}(\tilde{\mathcal{V}}) = \mathrm{clo}(\tilde{\mathcal{L}} \cap \mathcal{A}) \subseteq \tilde{\mathcal{L}} \cap \mathrm{clo}(\mathcal{A}) \subseteq \tilde{\mathcal{L}}$ we have $\alpha \in \tilde{\mathcal{L}}$, hence $\alpha \in \tilde{\mathcal{L}} \cap \mathcal{A}$.
"⊇" Pick $\alpha \in \tilde{\mathcal{L}} \cap \mathcal{A}$ and construct a sequence $(r_i)_{i \in \mathbb{N}}$ s.t. $r_i = \alpha|_{[0,i]}$. As $\alpha \in \tilde{\mathcal{L}}$ and $\tilde{\mathcal{L}}$ is associated to $\tilde{f}$ in (59) via (15), we know that $\alpha(i+1) \in \tilde{f}(r_i)$ for all $i \in \mathbb{N}$. Doing induction over i we see that $r_0 = \epsilon \in \mathrm{pfx}\, \tilde{\mathcal{V}}_t$ and for $r_i \in \mathrm{pfx}\, \tilde{\mathcal{V}}_t$ we have two cases. Either $\alpha(i+1) \notin \Sigma_{uc}$ implying $r_{i+1} \in \mathrm{pfx}\, \tilde{\mathcal{V}}_t$ from (59), or $\alpha(i+1) \in \Sigma_{uc}$ in which case condition (ii) from Definition 2 implies $r_{i+1} \in \mathrm{pfx}\, \tilde{\mathcal{V}}_t$. With this, it follows that $\alpha \in \mathrm{clo}(\tilde{\mathcal{V}}_t)$, hence $\alpha \in \mathrm{clo}(\tilde{\mathcal{V}}_t) \cap \mathcal{A}$. Using condition (i) in Definition 2 again we get $\alpha \in \tilde{\mathcal{V}}_t$. □

Given that (61) holds, the "⇒" part of (26) can be easily proven by contradiction using Lemma 7. I.e., given $t \in \mathrm{cfx}_{\mathcal{A}} \mathcal{G}$ we know that there exists $\tilde{\mathcal{V}}_t \subseteq \mathcal{G} \cap (t\Sigma^\omega)$ s.t. $\tilde{\mathcal{V}}_t$ fulfills properties (i) and (ii) in Definition 2. As $\mathcal{G} \subseteq \mathcal{A}$, we have $\tilde{\mathcal{V}}_t \subseteq \mathcal{A} \cap (t\Sigma^\omega)$ and we can therefore apply Lemma 7. Now assume that $\delta(q_0, t) \notin \mathsf{Win}(M)$ and therefore $\delta(q_0, t) \in \overline{Z}^\infty$. Then it follows from Lemma 7 and (61) that there exists a string $t\beta \in \tilde{\mathcal{V}}_t$ s.t. $s\beta \notin \mathcal{G}$. However, this is a contradiction to the hypothesis $\tilde{\mathcal{V}}_t \subseteq \mathcal{G} \cap (t\Sigma^\omega)$ and we can therefore conclude $\delta(q_0, t) \in \mathsf{Win}(M)$, what proves the statement.

C. Technical Propositions and Proofs to Support the Comparison

Reactive Synthesis via Supervisory Control

The following technical proposition summarises relevant properties obtained by our construction of $\mathcal{L}''$ in Eqs. (31), (32) and (36).

Proposition 3. Given non-empty alphabets U, Y, Σ and Σ_{uc}, where $\Sigma = U \dot\cup Y$ and $\Sigma_{\mathrm{uc}} = U$, consider any supervisor $f : \Sigma^* \to \Gamma$ with associated behaviour $\mathcal{L}$ that solves $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ for parameters $\mathcal{A}$, $\mathcal{G} \subseteq (UY)^\omega$ as plant and specification, respectively. Then the behaviour $\mathcal{L}''$ defined by Eqs. (31), (32) and (36), exhibits properties (RS1) – (RS3). Moreover, we have that $\mathcal{A}$ and $\mathcal{L}''$ do not deadlock and that $\mathcal{A} \cap \mathcal{L}'' \subseteq \mathcal{A} \cap \mathcal{L}$.

Proof. As a preliminary observation, we show the $\mathcal{L}'$ and $(UY)^\omega$ are non-conflicting. Pick any $s \in (\mathrm{pfx}\,\mathcal{L}') \cap (\mathrm{pfx}(UY)^\omega)$. If $s \in (UY)^*$, we refer to universal controllability (SC1) to obtain $su \in \mathrm{pfx}\,\mathcal{L}'$ for any $u \in U$. If $s \notin (UY)^*$, we must have $s \in (UY)^*U$ and we refer to Eq. (34) to obtain that $sy \in \mathrm{pfx}\,\mathcal{L}'$ for some $y \in Y$. In both cases, we have established the existence of some $\sigma \in \Sigma$ such that $s\sigma \in (\mathrm{pfx}\,\mathcal{L}') \cap (\mathrm{pfx}(UY)^\omega)$, i.e., the two languages do not deadlock. Recall that $\mathcal{L}'$ is topologically closed by (SC1), and note hat $(UY)^\omega$ is topologically closed, too. Thus, not to deadlock implies non-conflictingness.

We now turn to the properties (RS1)–(RS3). *Ad (RS1).* The intersection $\mathcal{L}''$ of two topologically closed languages is itself topologically closed. *Ad (RS2).* Pick any $s \in \Sigma^*$, u', $u'' \in U$ and assume that $su' \in \mathrm{pfx}\,\mathcal{L}''$. In partiular, $su' \in \mathrm{pfx}\,\mathcal{L}'$ and, by (SC2), $su'' \in \mathrm{pfx}\,\mathcal{L}'$. By $su' \in \mathrm{pfx}\,\mathcal{L}'' \subseteq \mathrm{pfx}(UY)^\omega$, we obtain $su'' \in \mathrm{pfx}(UY)^\omega$. Since $\mathcal{L}'$ and $(UY)^\omega$ are non-conflicting, this implies $su'' \in \mathrm{pfx}(\mathcal{L}'' \cap (UY)^\omega) = \mathrm{pfx}\,\mathcal{L}''$. *Ad (RS3)* Pick any $s \in \Sigma^*$, y', $y'' \in Y$ and assume that $sy' \in \mathrm{pfx}\,\mathcal{L}''$. $sy'' \in \mathrm{pfx}\,\mathcal{L}''$. This implies $sy' \in \mathrm{pfx}\,\mathcal{L}'$ and $sy'' \in \mathrm{pfx}\,\mathcal{L}'$, and, by Eq. (35), $y' = y''$.

To prove that $\mathcal{A} \cap \mathcal{L}'' \subseteq \mathcal{L}$, pick any $w \in \mathcal{A} \cap \mathcal{L}''$. Since $\mathcal{L}$ is topologically closed, it is suffient to show that $\mathrm{pfx}\,w \subseteq \mathrm{pfx}\,\mathcal{L}$ and we do so by induction. By $\mathcal{L} \neq \emptyset$, we have $\epsilon \in \mathrm{pfx}\,\mathcal{L}$, i.e., the claim is true for the prefix $\epsilon < w$ of length 0. Now consider $s \in \Sigma^*$, $\sigma \in \Sigma$ with $s\sigma < w$ and assume that $s \in \mathrm{pfx}\,\mathcal{L}$. If $s \in (UY)^*$, we have that $\sigma \in U$ and refer to (RS2) to obtain $s\sigma \subseteq \mathrm{pfx}\,\mathcal{L}$. If $s \notin (UY)^*$, we must have $s \in (UY)^*U$, and, hence, $\sigma \in f'(s) \cap Y$. Denote $Y_{\mathrm{f}}(s)$ the ouputs that comply with the original supervisor f given s, i.e., $Y_{\mathrm{f}}(s) := f(s) \cap Y = \{\, y \in Y \mid sy \in \mathrm{pfx}\,\mathcal{L} \,\}$. Likewise, denote Y_{a} the ouputs that comply with the plant given s, i.e., $Y_{\mathrm{a}}(s) := \{\, y \in Y \mid sy \in \mathrm{pfx}\,\mathcal{A} \,\}$. Then non-conflictingness of $\mathcal{A}$ and $\mathcal{L}$ implies $Y_{\mathrm{a}}(s) \cap Y_{\mathrm{f}}(s) \neq \emptyset$. By the definition of the post-processed supervisor h, we have $h(s) \cap Y = (Y_{\mathrm{a}}(s) \cap Y_{\mathrm{f}}(s))$. By the definition of f', we have that $f'(s) \cap Y$ is a singleton, i.e., we obtain that $\{\sigma\} = f'(s) \cap Y \subseteq h(s) \cap Y = Y_{\mathrm{a}}(s) \cap Y_{\mathrm{f}}(s)$. In particular, this implies $\sigma \in f(s)$. We extend $s\sigma$ by an arbitrary $\beta \in \Sigma_{\mathrm{uc}}^\omega$, and refer to the induction hypothesis $s \in \mathrm{pfx}\,\mathcal{L}$ to obtain $s\sigma\beta \in \mathcal{L}$. This implies $s\sigma \in \mathrm{pfx}\,\mathcal{L}$ and thereby completes the induction step.

To see that $\mathcal{A}$ and $\mathcal{L}''$ do not deadlock, pick any $s \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L}'')$. If $s \in (UY)^*$, we refer to (RS2) to obtain $sU \subseteq \mathrm{pfx}\,\mathcal{L}''$. Since $\mathcal{A} \subseteq (UY)^\omega$, there must exist some $u \in U$ such that $su \in \mathrm{pfx}\,\mathcal{A}$. Thus, we have $su \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L}'')$. If $s \notin (UY)^*$, we must have $s \in (UY)^*U$. Consider the same sets of enabled output events as above, i.e., $Y_{\mathrm{f}} := f(s) \cap Y = \{\, y \in Y \mid sy \in \mathrm{pfx}\,\mathcal{L} \,\}$ and $Y_{\mathrm{a}} := \{\, y \in Y \mid sy \in \mathrm{pfx}\,\mathcal{A} \,\}$, and we recall that $Y_{\mathrm{a}} \cap Y_{\mathrm{f}} \neq \emptyset$. By the definition of the post-processed supervisor h, we have $h(s) = \Sigma_{\mathrm{uc}} \cup (Y_{\mathrm{a}} \cap Y_{\mathrm{f}})$. Thus, $y \in f'(s)$ for some $y \in Y_{\mathrm{a}} \cap Y_{\mathrm{f}}$. As above, we extend sy by an arbitrary $\beta \in \Sigma_{\mathrm{uc}}^\omega$ to obtain $sy\beta \in \mathcal{L}'$, and, hence $sy \in \mathrm{pfx}\,\mathcal{L}'$. Since $\mathcal{L}'$ and $(UY)^\omega$ are non-conflicting, we finally obtain that $sy \in (\mathrm{pfx}\,\mathcal{L}') \cap (\mathrm{pfx}(UY)^\omega) = \mathrm{pfx}\,\mathcal{L}''$. □

Proof of Theorem 2. We refer to the above proposition an obtain that $\mathcal{L}''$ satisfies (RS1) – (RS3) and that $\mathcal{A} \cap \mathcal{L}'' \subseteq \mathcal{A} \cap \mathcal{L}$. Hence $\mathcal{L}'' \subseteq (\mathcal{A} \cap \mathcal{L}) \cup (\Sigma^\omega - \mathcal{A})$. Since

$\mathcal{L}$ solves the supervisory control problem, Problem 2, we also have that $\mathcal{A} \cap \mathcal{L} \subseteq \mathcal{G}$. Thus, $\mathcal{L}'' \subseteq \mathcal{G} \cup (\Sigma^\omega - \mathcal{A}) = \mathcal{A} \to \mathcal{G}$. This establishes that $\mathcal{L}''$ solves Problem 1. Regarding ω-regularity, we first observe that all post-processing is performed for topologically closed languages. Thus, there are no acceptance conditions. The language associated with g is the intersection of $\operatorname{clo} A$ and $\mathcal{L}$ and amounts to a product composition. This concludes the proof of Theorem 2. □

Supervisory Control via Reactive Synthesis (Control-Patterns as Outputs)

Our proof of Theorem 3 is based on the following two technical propositions.

Proposition 4. Consider a finite alphabet Σ, uncontrollable events Σ_{uc}, $\emptyset \neq \Sigma_{\mathrm{uc}} \subseteq \Sigma$, control patterns $\Gamma := \{\, \gamma \subseteq \Sigma \,|\, \Sigma_{\mathrm{uc}} \subseteq \gamma \,\}$ and extended variants thereof as defined in Eq. (38), and, for an arbitrary language $\mathcal{L}' \subseteq (\Sigma'\Gamma')^\omega$, the variant $\mathcal{L} \subseteq \Sigma^\omega$ defined in Eq. (41). If $\mathcal{L}'$ is non-empty and satisfies (RM1) and (RM2) with $U = \Sigma'$ and output $Y = \Gamma'$, then $\mathcal{L}$ is non-empty and satisfies (SC1) and (SC2). Consider the additional parameters $\mathcal{A} \subseteq \Sigma^\omega$ and $\mathcal{G} \subseteq \Sigma^\omega$ and extended variants thereof as defined in Eqs. (39) and (40). If $\mathcal{L}' \subseteq \mathcal{A}' \to \mathcal{G}'$ then $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$.

Proof. In general, topological closedness is not retained under projection. Thus, in order to establish (SC1) by (RM1) we need to explicitly refer to the special situation of alternating visible symbols and invisible symbols, as well as to the fact that the alphabet under consideration is finite. Pick an arbitrary $\beta \in \operatorname{clo} \mathcal{L}$ and denote $(s_i)_{i\in\mathbb{N}}$ a sequence $s_i < s_{i+1} \in \operatorname{pfx} \beta \subseteq \operatorname{pfx} \mathcal{L}$ for all $i \in \mathbb{N}$. Define the sets of $*$-words from the full alphabet that comply with s_i, $i \in \mathbb{N}$, by

$$\begin{aligned} R_i := \{\, r \in (\Sigma'\Gamma')^* \,|\, &\exists\, \alpha \in (\Sigma'\Gamma')^\omega \text{ with } r\alpha \in \mathcal{L}' \text{ such that} \\ &\mathrm{p}_{\Sigma'}\, r = 0 s_i \quad \text{and} \quad \forall\, \gamma \in \Gamma',\, \sigma \in \Sigma',\, s\gamma\sigma \in \operatorname{pfx} r :\ \sigma \in \gamma \,\}\,. \end{aligned} \tag{62}$$

Note that $s_i \in \operatorname{pfx} \mathcal{L}$ implies that $R_i \neq \emptyset$, and observe by $\mathrm{p}_{\Sigma'}\, R_i = \{0 s_i\}$ and $R_i \subseteq (\Sigma'\Gamma')^*$ that words in R_i have uniform length. In particular, R_i is a finite set. Moreover, we have the following property:

$$\forall\, i,\ j \in \mathbb{N},\ i \leq j,\ t \in R_j,\ \exists\, r \in R_i :\ r \leq t\,; \tag{63}$$

i.e., R_j consists of specific postfixes from R_i. Next, denote

$$R_{i,j} := \{\, r \in R_i \,|\, \exists\, t \in R_j :\ r \leq t \,\}\,, \tag{64}$$

where $i,\ j \in \mathbb{N},\ i \leq j$. By Eq. (63), $R_{i,j}$ is monotonously decreasing w.r.t. j, and, since R_j is non-empty, so is $R_{i,j}$, i.e., $\emptyset \neq R_{i,j+1} \subseteq R_{i,j}$ for all $i,\ j \in \mathbb{N},\ i \leq j$. For $i \in \mathbb{N}$, consider the limit

$$N_i := \bigcap_{j \geq i} R_{i,j} \ \subseteq\ R_i \tag{65}$$

and observe that $N_i \neq \emptyset$, since all components of the monotone sequence are non-empty and finite. To establish that

$$\forall\, i \in \mathbb{N},\ r \in N_i\ \exists\, t \in N_{i+1} :\ r \leq t \tag{66}$$

by contradiction, assume that we can pick $i \in \mathbb{N},\ r \in N_i$ such that $rv \notin N_{i+1}$ for all $v \in \Sigma^*$. Since the words in R_i and R_{i+1} are of uniform length, the last clause is equivalent to $rv \notin N_{i+1}$ for all $v \in \Sigma^l$ and some suitably chosen $l \in \mathbb{N}$. Next, referring to the definition of N_{i+1}, we pick for each $v \in \Sigma^l$ some $j_v \geq i+1$ such that

$rv \notin R_{i+1,j_v}$ and denote the maximum $k := \max_{v\in\Sigma^l} j_v \in \mathbb{N}$. By monotonicity of $R_{i+1,j}$ w.r.t. j, this implies $rv \notin R_{i+1,k}$ for all $v \in \Sigma^l$, and, hence, $rvw \notin R_k$ for all $v \in \Sigma^l$, $w \in \Sigma^*$. The latter collapses to $rt \notin R_k$ for all $t \in \Sigma^*$ and thereby implies $r \notin R_{i,k}$. This is a contradiction with $r \in N_i$ and concludes the proof of Eq. (66). By the latter property, we begin with an arbitrary $r_1 \in N_1$ and successively obtain a a sequence $(r_i)_{i\in\mathbb{N}}$, $r_i \in N_i$, $r_i < r_{i+1}$ for all $i \in \mathbb{N}$. Denote the singleton limit by $\alpha \in (\Sigma'\Gamma')^\omega$. We then have $\mathrm{p}_{\Sigma'}\,\alpha = 0\beta$ and, by topological closedness (RM1), $\alpha \in \mathcal{L}'$. Referring to the second condition in Eq. (62), we also obtain that $\sigma \in \gamma$ for all $\gamma \in \Gamma'$, $\sigma \in \Sigma'$ and s with $s\gamma\sigma \in \operatorname{pfx}\alpha$, and, hence $\beta \in \mathcal{L}$. This concludes the proof of topological closedness of $\mathcal{L}$ and therefore establishes (SC1).

In order to establish controllability (SC2), pick any $s \in \operatorname{pfx}\mathcal{L}$ and $\sigma \in \Sigma_{\mathrm{uc}}$ and choose $\beta \in \Sigma^\omega$ such that $s\beta \in \mathcal{L}$. By the definition Eq (41) of $\mathcal{L}$, we can further choose $r_1 \in (\Sigma'\Gamma')^*$ and $\alpha \in (\Sigma'\Gamma')^\omega$ with $r_1\alpha \in \mathcal{L}'$, such that $\mathrm{p}_{\Sigma'}\,r_1 = 0s$ and $\sigma_1 \in \gamma_1$ for all $v_1 \in (\Sigma'\Gamma')^*$ with $v_1\gamma_1\sigma_1 \leq r_1$. Referring to (RS2), we extend $r_1 \operatorname{pfx}\mathcal{L}'$ by σ to obtain $r_1\sigma \in \operatorname{pfx}\mathcal{L}'$. Writing $r_1 = v_1\gamma_1$ with $\gamma_1 \in \Gamma'$ we observe $\sigma \in \gamma_1$ by $\sigma \in \Sigma_{\mathrm{uc}}$. Referring to $\mathcal{L}' \subseteq (\Sigma'\Gamma')^\omega$, we further extend $r_1\sigma$ by some $\gamma \in \Gamma'$ to obtain $r_2 := r_1\sigma\gamma \in \operatorname{pfx}\mathcal{L}'$ and note that and $\sigma_2 \in \gamma_2$ for all $v_2 \in (\Sigma'\Gamma')^*$ with $v_2\gamma_2\sigma_2 \leq r_2$. This construction is repeated to obtain a strictly monotone sequence $(r_i)_{i\in\mathbb{N}}$, $r_i \in (0\Gamma')(\Sigma\Gamma')^*$, $r_i < r_{i+1}$ for all $i \in \mathbb{N}$, and we denote the singleton limit $\alpha' \in (0\Gamma')(\Sigma\Gamma')^\omega$. In particular, $\operatorname{pfx}\alpha' \subseteq \operatorname{pfx}\mathcal{L}'$ and we obtain $\alpha' \in \mathcal{L}'$ by topological closedness (RM1). By our construction, we have that $\sigma' \in \gamma'$ for all $v' \in (\Sigma'\Gamma')^*$ with $v'\gamma'\sigma' < \alpha'$. Let $\beta' := \mathrm{p}_\Sigma\,\alpha$ to obtain $0\beta' = \mathrm{p}_{\Sigma'}\,\alpha$ and, thus, $\beta' \in \mathcal{L}$. In particular, $0s\sigma = \mathrm{p}_{\Sigma'}\,r_2 \in 0\operatorname{pfx}\beta'$ and, hence, $s\sigma \in \operatorname{pfx}\mathcal{L}$.

For non-emptyness of $\mathcal{L}$, we refer to $\mathcal{L}' \neq \emptyset$ and (RM2) to observe that $0 \in \operatorname{pfx}\mathcal{L}'$. Thus, we can pick $r_1 \in 0\Gamma'$ such that $r_1 \in \operatorname{pfx}\mathcal{L}$. Now assume that $r_i \in \operatorname{pfx}\mathcal{L}' \cap (0\Gamma')(\Sigma\Gamma')^*$ for some $i \in \mathbb{N}$. Then, we can write $r_i = s\gamma$ for some $\gamma \in \Gamma'$. Because $\Sigma_{\mathrm{uc}} \subseteq \gamma$, we can extend $s\gamma$ by some $\sigma \in \gamma \cap \Sigma$ to observe $s\gamma\sigma \in \operatorname{pfx}\mathcal{L}'$ by (RM2). Hence, there exists $r_{i+1} \in r_i\sigma\Gamma'$ such that $r_{i+1} \in \operatorname{pfx}\mathcal{L}' \cap (0\Gamma')(\Sigma\Gamma')^*$. This establishes a monotone sequence $(r_i)_{i\in\mathbb{N}}$, $r_i < r_{i+1} \in \operatorname{pfx}\mathcal{L}'$ for all $i \in \mathbb{N}$. We denote the singleton limit $\alpha \in (0\Gamma')(\Sigma'\Gamma')^\omega$ and conclude $\alpha \in \mathcal{L}'$ by topological closedness (RM1). Moreover, there exists $\beta \in \Sigma^\omega$ such that $0\beta = \mathrm{p}_{\Sigma'}\,\alpha$. By construction, we also have $\sigma \in \gamma$ for all $\sigma \in \Sigma'$, $\gamma \in \Gamma'$, $s\gamma\sigma < \alpha$, and, hence, $\beta \in \mathcal{L}$.

Regarding the inclusion $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$, pick any $\beta \in \mathcal{L}$ and choose $\alpha \in \mathcal{L}'$ according to Eq (41), i.e, $\mathrm{p}_{\Sigma'}\,\alpha = 0\beta$ and $\sigma \in \gamma$ for all $s \in (\Sigma'\Gamma')^*$ with $s\gamma\sigma < \alpha$. If $\beta \in \mathcal{A}$, we have $\alpha \in \mathcal{A}'$ by Eq. (39). Since $\mathcal{L}'$ solves Problem 1, this implies $\alpha \in \mathcal{G}'$, and, by Eq (40), $0\beta = \mathrm{p}_{\Sigma'}\,\alpha \in 0\mathcal{G}$. Thus, we observe that $\beta \in \mathcal{G}$ and conclude the proof of $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$. □

Proposition 5. Consider a finite alphabet Σ, uncontrollable events Σ_{uc}, $\emptyset \neq \Sigma_{\mathrm{uc}} \subseteq \Sigma$, control patterns $\Gamma := \{\,\gamma \subseteq \Sigma \,|\, \Sigma_{\mathrm{uc}} \subseteq \gamma\,\}$ and extended variants Σ' and Γ' thereof as defined in Eq. (38). Given a plant $\mathcal{A} \subseteq \Sigma^\omega$, construct the assumption $\mathcal{A}' \subseteq (\Sigma'\Gamma')^\omega$ by Eq. (40). For a reactive module $\mathcal{L}' \subseteq (UY)^\omega$ with input range $U := \Sigma'$, output range $Y := \Gamma'$ and satisfying (RS1) and (RS2), consider the supervisor candidate $\mathcal{L} \subseteq \Sigma^\omega$ defined in Eq. (41). If $\operatorname{pfx}\mathcal{A}'$ in interconnection with $\operatorname{pfx}\mathcal{L}'$ does not deadlock, then neither does $\operatorname{pfx}\mathcal{A}$ in interconnection with $\operatorname{pfx}\mathcal{L}$.

Proof. To establish the absence of deadlocks, we pick an arbitrary $s \in (\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L})$ and extend this synchronously by one more symbol.

We first refer to $s \in \operatorname{pfx}\mathcal{L}$ and extend by $\beta \in \Sigma^\omega$ to obtain $s\beta \in \mathcal{L}$. By the definition of $\mathcal{L}$ in Eq. (41), we can choose $\alpha \in \mathcal{L}'$ such that $\mathrm{p}_{\Sigma'}\,\alpha = 0s\beta$ and such

that all prefixes comply with all past control patterns, i.e., we have $\operatorname{pfx}\alpha \subseteq L_{\mathrm{SCT}}$, where

$$L_{\mathrm{SCT}} := \operatorname{pfx}\{\, t \in (\Sigma'\Gamma)^* \,|\, \forall\, \sigma \in \Sigma,\, \gamma \in \Gamma, r \in \operatorname{pfx} t :\ r\gamma\sigma \leq t \ \Rightarrow\ \sigma \in \gamma \,\}\,. \tag{67}$$

In particular, we can rewrite

$$\alpha = 0\gamma_1\sigma_1\gamma_2\sigma_2 \cdots \gamma_n\sigma_n\gamma_{n+1}\sigma_{n+1} \cdots\ , \tag{68}$$

with $\sigma_k \in \gamma_k \subseteq \Gamma$ for $k \in \mathbb{N}$ to observe that $s = \sigma_1\sigma_2 \cdots \sigma_n$, $n := |s|$. We now refer to $s \in \operatorname{pfx}\mathcal{A}$ and extend by $\beta' \in \Sigma^\omega$ to obtain $s\beta' \in \mathcal{A}$. Here, we write $\beta' = \sigma'_{n+1}\sigma'_{n+2}\sigma'_{n+3} \cdots$ and let $\gamma'_k := \Gamma$ for all $k \in \mathbb{N}$ to assemble

$$\alpha' := 0\gamma_1\sigma_1\gamma_2\sigma_2 \cdots \gamma_n\sigma_n\gamma'_{n+1}\sigma'_{n+1} \cdots\ . \tag{69}$$

Again, we have $\operatorname{pfx}\alpha' \subseteq L_{\mathrm{SCT}}$ and, by $s\beta' \in \mathcal{A}$, we conclude $\alpha' \in \mathcal{A}'$.

Both ω-words α and α' share the prefix $0t$ with $t := \gamma_1\sigma_1\gamma_2\sigma_2 \cdots \gamma_n\sigma_n$ and we have that $0t \in (\operatorname{pfx}\mathcal{A}') \cap (\operatorname{pfx}\mathcal{L}')$. Since $\mathcal{A}'$ and $\mathcal{L}'$ do not deadlock, we can extend $0t$ by $\sigma'' \in \gamma'' \subseteq \Gamma$ to obtain $0t\gamma''\sigma'' \in (\operatorname{pfx}\mathcal{A}') \cap (\operatorname{pfx}\mathcal{L}')$. Extending $0t\gamma''\sigma''$ by α'' to obtain $0t\gamma''\sigma''\alpha'' \in \mathcal{A}'$ we observe $0s\sigma''\,\mathrm{p}_{\Sigma'}\,\alpha'' = \mathrm{p}_{\Sigma'}(0t\gamma''\sigma''\alpha'') \in 0\mathcal{A}$, and, hence, $s\sigma'' \in \operatorname{pfx}\mathcal{A}$. We can also extend $0t\gamma''\sigma''$ to an ω-word in $\mathcal{L}'$, however, in order to address Eq. (41) we need to take care that all prefixes comply with past control patterns. For an inductive argument, pick any $r\sigma \in \operatorname{pfx}\mathcal{L}'$. By the alternating structure of $\mathcal{L}'$, we find $\gamma \in \Gamma$ such that $r\sigma\gamma \in \operatorname{pfx}\mathcal{L}'$ and, by the free input (RS2), we can choose any $\varrho \in \gamma$ to obtain $r\sigma\gamma\varrho \in \operatorname{pfx}\mathcal{L}'$. This construction maintains compliance with control patterns, i.e., $r\sigma \in L_{\mathrm{SCT}}$ implies $r\sigma\gamma\varrho \in L_{\mathrm{SCT}}$. Applying this construction to the initial string $0t\gamma''\sigma'' \in \operatorname{pfx}\mathcal{L}'$, and referring to topological closedness (RS1), we obtain an ω-word $0t\gamma''\sigma''\alpha''' \in \mathcal{L}'$ such that $\operatorname{pfx}(0t\gamma''\sigma''\alpha''') \subseteq L_{\mathrm{SCT}}$. For the projection, we have $\mathrm{p}_{\Sigma'}(0t\gamma''\sigma''\alpha''') = 0s\sigma''\,\mathrm{p}_{\Sigma'}\,\alpha$, and, referring to the definition of $\mathcal{L}$ in Eq. (41), we obtain $s\sigma'' \in \operatorname{pfx}\mathcal{L}$. This concludes the proof. □

Proof of Theorem 3. Regarding $\mathcal{L}$, non-emptyness, (SC1) and (SC2) are established by Proposition 4. Moreover, we obtain by Proposition 5 that $\mathcal{A}$ and $\mathcal{L}$ do not deadlock. Since both languages are topologically closed, this implies non-conflictingness. In turn, non-emptyness of $\mathcal{A}$ and $\mathcal{L}$ implies $\emptyset \neq \mathcal{K} := \mathcal{A} \cap \mathcal{L}$. For the specification, we again refer to Proposition 4 to obtain $\mathcal{K} = \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{A} \cap (\mathcal{A} \to \mathcal{G}) = \mathcal{A} \cap \mathcal{G} \subseteq \mathcal{G}$. Now assume that $\mathcal{L}'$ is ω-regular and consider a finite automaton realisation $A' = (Q,\, \Sigma' \cup \Gamma',\, \delta,\, Q_{\mathrm{o}})$. Since $\mathcal{L}'$ is topologically closed, we do not need to consider an acceptance condition and we can without loss of generality assume that A is deterministic and reachable. A realisation of $\mathcal{L}$ can be obtained in three stages. First, we can test on a per state basis whether or not enabled transition are labeled in compliance with the recent control pattern. This step may require to split states in order for the most recent control pattern to be unique for each state. The resulting automaton A'' then realises the language

$$\mathcal{L}'' := \{\alpha \in (\Sigma'\Gamma')^\omega \,|\, \forall\, \gamma \in \Gamma',\, \sigma \in \Sigma',\, s\gamma\sigma \in \operatorname{pfx}\alpha :\ \sigma \in \gamma \,\}\,. \tag{70}$$

At a second stage, we refer to well known algorithms that implement the projection to obtain A''' to realise $\mathcal{L}''' := \mathrm{p}_{\Sigma'}\,\mathcal{L}''$ and observe that

$$\begin{aligned}\mathcal{L}''' := \{\beta \in \Sigma^\omega \,|\, &\exists\, \alpha \in \mathcal{L}' \text{ with}\\ &\beta = \mathrm{p}_{\Sigma'}\,\alpha \quad \text{and} \quad \forall\, \gamma \in \Gamma',\, \sigma \in \Sigma',\, s\gamma\sigma \in \operatorname{pfx}\alpha :\ \sigma \in \gamma \,\}\,.\end{aligned} \tag{71}$$

Although in general, projection does not retain determinism, in the absence of an acceptance condition we may determinise the result by the common subset-construction. Third, we perform the intersection with $0(\Gamma'\Sigma')^\omega$ and drop the leading 0-symbol. Again, these operations retain regularity and respective algorithms are well known. □

The following proposition is used to derive Corollary 1 from Theorem 3.

Proposition 6. Let Σ be a finite alphabet with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$ and let $\mathcal{L}$ be the solution of $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ with a non-empty topologically-closed plant $\mathcal{A} \subseteq \Sigma^\omega$ and an upper-bound specification $\mathcal{G} \subseteq \Sigma^\omega$. Furthermore, let

$$\mathcal{L}' := \{\, \alpha \in (\Sigma'\Gamma')^\omega \mid \forall\, \gamma \in \Gamma',\, s \in (\Sigma'\Gamma')^* :\ s\gamma < \alpha \to \gamma = f(\mathrm{p}_\Sigma\, s) \cup \{0\} \,\}, \quad (72)$$

where f denotes the supervisor with behaviour $\mathcal{L}$. Then $\mathcal{L}'$ solves $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$ with $U = \Sigma'$, $Y = \Gamma'$, $\mathcal{A}'$ and $\mathcal{G}'$ defined via Eqs. (38)-(40).

Proof. We need to prove that (RS1)-(RS3) holds for $\mathcal{L}'$, $\mathcal{L}'$ and $\mathcal{A}'$ do not deadlock, $\mathcal{L}' \neq \emptyset$ and $\mathcal{L}' \subseteq \mathcal{A}' \to \mathcal{G}'$. As the last statement is implied by $\mathcal{L}' \cap \mathcal{A}' \subseteq \mathcal{G}'$, we prove this statement instead.
▶ (RS1): Pick any $\alpha \in \mathrm{clo}\,\mathcal{L}'$ and $\gamma \in \Gamma'$, $s \in (\Sigma'\Gamma')^*$ such that $s\gamma < \alpha$. By $\alpha \in \mathrm{clo}\,\mathcal{L}'$, we can choose an arbitrary length prefix $t < \alpha$ such that $t \in \mathrm{pfx}\,\mathcal{L}'$, i.e., such that there exists an extension β with $t\beta \in \mathcal{L}'$. In particular, we can take the choice such that $s\gamma < t < t\beta$, which implies $\gamma = f(\mathrm{p}_\Sigma\, s)$. We conclude that $\alpha \in \mathcal{L}'$ and obtain that $\mathcal{L}'$ is closed.
▶ (RS2): Pick an arbitary ω-word $\beta = \sigma_1\sigma_2\sigma_3 \cdots \in \Sigma'^\omega$ and denote $\gamma_i := f(\sigma_1\sigma_2 \cdots \sigma_i) \cup \{0\} \in \Gamma'$. Then $\alpha := \sigma_1\gamma_1\sigma_2\gamma_2 \cdots \in (\Sigma'\Gamma')^\omega$ qualifies for $\alpha \in \mathcal{L}'$. This establishes that $\mathrm{p}_{\Sigma'}\, \mathcal{L}' = \Sigma'^\omega$ which in turn implies a locally free input Σ'.
▶ (RS3): Consider some s such that $s\gamma \in \mathrm{pfx}\,\mathcal{L}'$ and $s\gamma' \in \mathrm{pfx}\,\mathcal{L}'$ for $\gamma,\ \gamma' \in \Gamma'$. By the definitiom of $\mathcal{L}'$ this immediately implies $\gamma = f(\mathrm{p}_\Sigma\, s) \cup \{0\} = \gamma'$ and concludes the proof of (RM3).
▶ $\emptyset \neq \mathcal{L}'$: To see this, we use the same witness $\alpha \in \mathcal{L}'$ as in the proof of (RM2).
▶ $\mathcal{A}' \cap \mathcal{L}' \subseteq \mathcal{G}'$: Pick any $\alpha \in \mathcal{A}' \cap \mathcal{L}'$. From $\alpha \in \mathcal{A}'$ and Eq. (38) we obtain (i) that $\mathrm{p}_{\Sigma'}\, \alpha \in 0\mathcal{A}$ and (ii), for all $\gamma \in \Gamma'$, $\sigma \in \Sigma'$, $s\gamma\sigma \in \mathrm{pfx}\,\alpha$, that $\sigma \in \gamma$. From $\alpha \in \mathcal{L}'$, we obtain that (iii) $\sigma \in \gamma = f(\mathrm{p}_\Sigma\, s) \cup \{0\}$ with the same quantification as in (ii). Referring to the alternating structure $\alpha \in \mathcal{A}' \subseteq 0(\Gamma'\Sigma)^\omega$, we cancel γ and obtain, for all $\sigma \in \Sigma$ and $r\sigma \in \mathrm{pfx}\,\mathrm{p}_\Sigma\,\alpha$, that $\sigma \in f(r)$. This implies $\mathrm{p}_\Sigma\,\alpha \in \mathcal{L}$, and, hence $\mathrm{p}_\Sigma\,\alpha \in \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{G}$. Finally, we obtain $\mathrm{p}_{\Sigma'}\,\alpha = 0\,\mathrm{p}_\Sigma\,\alpha \in 0\mathcal{G}$, which establishes $\alpha \in \mathcal{G}'$ by Eq. (39) and we observe $\mathcal{L}' \subseteq \mathcal{A}' \to \mathcal{G}'$ from $\mathcal{A}' \cap \mathcal{L}' \subseteq \mathcal{G}'$.
▶ $\mathcal{A}'$ and $\mathcal{L}'$ do not deadlock: Pick any $s \in (\mathrm{pfx}\,\mathcal{A}') \cap (\mathrm{pfx}\,\mathcal{L}')$. First, assume that $s \in (\Sigma'\Gamma')^*$. Here, we choose $\sigma \in \Sigma'$ such that $s\sigma \in \mathrm{pfx}\,\mathcal{A}'$ and to refer to (RM2) for $s\sigma \in \mathrm{pfx}\,\mathcal{L}'$. For the second case, we have $s \in 0((\Gamma'\Sigma')^*)$. From $s \in (\mathrm{pfx}\,\mathcal{A}')$ and Eq. (38) we obtain (i) that $\mathrm{p}_{\Sigma'}\, s \in \mathrm{pfx}(0\mathcal{A})$ and (ii), for all $\gamma \in \Gamma'$, $\sigma \in \Sigma'$, $t\gamma\sigma \in \mathrm{pfx}\,s$, that $\sigma \in \gamma$. From $s \in \mathrm{pfx}\,\mathcal{L}'$, we obtain that (iii) $\sigma \in \gamma = f(\mathrm{p}_\Sigma\, t) \cup \{0\}$ with the same quantification as in (ii). Referring to the alternating structure $s \in \mathrm{pfx}\,\mathcal{A}' \subseteq 0\,\mathrm{pfx}((\Gamma\Sigma)^*)$, we cancel γ and obtain, for all $\sigma \in \Sigma$ and $r\sigma \in \mathrm{pfx}\,\mathrm{p}_\Sigma\, s$, that $\sigma \in f(r)$. This implies $\mathrm{p}_\Sigma\, s \in \mathrm{pfx}\,\mathcal{L}$, and, hence $\mathrm{p}_\Sigma\, s \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L})$. Now we extend by $\sigma \in \Sigma$ for $\mathrm{p}_\Sigma\, s\sigma \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L})$. The second clause implies the existence of $\gamma \in \Gamma$ such that $\sigma \in \gamma = f(\mathrm{p}_\Sigma\, s)$. Hence, with $\gamma' := \gamma \cup \{0\}$, we have that $s\gamma'\sigma \in \mathrm{pfx}\,\mathcal{A}'$ and $\sigma \in \gamma' = f(\mathrm{p}_\Sigma\, s) \cup \{0\}$. This implies $s\gamma' \in \mathrm{pfx}\,\mathcal{L}'$ to conclude the proof. □

Supervisory Control via Reactive Synthesis (Input-Output Behaviours)

Proposition 7. Consider a finite alphabet Σ, uncontrollable events Σ_{uc}, $\emptyset \neq \Sigma_{\mathrm{uc}} \subsetneq \Sigma$, let $U = \Sigma_{\mathrm{uc}}$ and $Y = \Sigma - \Sigma_{\mathrm{uc}}$. Consider arbitrary languages $\mathcal{A}$, $\mathcal{L} \subseteq (UY)^\omega$ and the variant $\mathcal{L}' \subseteq \Sigma^\omega$ defined in Eq. (42). If $\mathcal{L}$ satisfies (RM1) and (RM2) and if $\operatorname{pfx}\mathcal{L}$ and $\operatorname{pfx}\mathcal{A}$ do not deadlock, then $\mathcal{L}'$ satisfies (SC1) and (SC2). Moreover, $\mathcal{L}' \cap (UY)^\omega = \mathcal{L}$, $(\operatorname{pfx}\mathcal{L}') \cap (\operatorname{pfx}((UY)^*)) = \operatorname{pfx}\mathcal{L}$. Finally, $\operatorname{pfx}\mathcal{L}'$ and $\operatorname{pfx}\mathcal{A}$ do not deadlock.

Proof. For closedness (SC1) pick any $\alpha \in \operatorname{clo}\mathcal{L}'$, i.e., $\operatorname{pfx}\alpha \subseteq \operatorname{pfx}\mathcal{L}'$. Note that

$$\operatorname{pfx}\mathcal{L}' = (\operatorname{pfx}\mathcal{L}) \cup ((\operatorname{pfx}\mathcal{L})\Sigma_{\mathrm{uc}}^*) \tag{73}$$

and that α must have infinitely many prefixes in at least one of the union components. We distinguish two cases. If, case (a), α has infinitely many prefixes in $\operatorname{pfx}\mathcal{L}$, we refer to refer to topological closedness (RM1) to obtain $\alpha \in \mathcal{L} \subseteq \mathcal{L}'$. For the complementary case (b), α has finitely many prefixes in $\operatorname{pfx}\mathcal{L}$ and, hence, infinitely many in $(\operatorname{pfx}\mathcal{L})\Sigma_{\mathrm{uc}}^*$. In particular, there is a longest prefix $s < \alpha$ with $s \in \operatorname{pfx}\mathcal{L}$). Therefore, we have $\alpha \in s\Sigma_{\mathrm{uc}}^\omega \subseteq (\operatorname{pfx}\mathcal{L})(\Sigma_{\mathrm{uc}}^\omega)$. In both cases, we have established $\alpha \in \mathcal{L}'$. Hence, $\mathcal{L}'$ is topologically closed. For controllability (SC2), pick any $s \in \operatorname{pfx}\mathcal{L}'$ and $\sigma \in \Sigma_{\mathrm{uc}}$, to obtain $s\sigma \in \operatorname{pfx}\mathcal{L}'$ by Eq. (73). The equality $\mathcal{L}' \cap (UY)^\omega = \mathcal{L}$ is immediate from the fact that for every $\alpha \in \mathcal{L}' - \mathcal{L}$ we have $\alpha \in (\operatorname{pfx}\mathcal{L})\Sigma_{\mathrm{uc}}^\omega$ and, hence, $\alpha \notin (UY)^\omega$. For the last equality in the proposition, we first show that $\mathcal{L}'$ and $(UY)^\omega$ are non-conflicting. Pick any $s \in (\operatorname{pfx}\mathcal{L}') \cap (\operatorname{pfx}(UY)^*)$. If $s \in (UY)^*$, we refer to (SC2) and extend s by any $\sigma \in U = \Sigma_{\mathrm{uc}}$ to obtain $s\sigma \in \operatorname{pfx}\mathcal{L}'$. Else, if $s \notin (UY)^*$, we must have $s \in (UY)^*U$. We write $s = tu$ with $u \in U$ and refer Eq. (73) for $t \in \operatorname{pfx}\mathcal{L}$, to (RM2) for $s = tu \in \operatorname{pfx}\mathcal{L}$. Thus, we can extend s by $\sigma \in Y$ such that $s\sigma = tu\sigma \in \operatorname{pfx}\mathcal{L} \subseteq \operatorname{pfx}\mathcal{L}'$. In both cases, we have established $s\sigma \in (\operatorname{pfx}\mathcal{L}') \cap (\operatorname{pfx}(UY)^*)$. Since both languages are topologically closed, this concludes the proof of non-conflictingness of $\mathcal{L}'$ and $(UY)^\omega$, which in turn implies $(\operatorname{pfx}\mathcal{L}') \cap (\operatorname{pfx}((UY)^*)) = \operatorname{pfx}(\mathcal{L}' \cap ((UY)^\omega)) = \operatorname{pfx}\mathcal{L}$. We are left to show that $\mathcal{A}$ and $\mathcal{L}'$ do not deadlock: $(\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L}') = (\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L}') \cap (\operatorname{pfx}((UY)^\omega)) = (\operatorname{pfx}\mathcal{A}) \cap (\operatorname{pfx}\mathcal{L})$; as $\mathcal{A}$ and $\mathcal{L}$ do not deadlock, the latter equality implies that $\mathcal{A}$ and $\mathcal{L}'$ do not deadlock, either. □

Proof of Theorem 4. Referring to Proposition 7, we have (SC1) and (SC2) for $\mathcal{L}'$. Note also that $\mathcal{L}' \neq \emptyset$ is an immediate consequence of non-emptyness of $\mathcal{L}$. Regarding the specification, we observe for the closed-loop behaviour $\mathcal{A} \cap \mathcal{L}' = \mathcal{A} \cap \mathcal{L}' \cap (UY)^\omega = \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{A} \cap (\mathcal{A} \to \mathcal{G}) = \mathcal{G}$. Again referring to Proposition 7, we also have that $\operatorname{pfx}\mathcal{L}'$ and $\operatorname{pfx}\mathcal{A}$ do not deadlock. Thus, non-conflictingness is implied by both $\mathcal{A}$ and $\mathcal{L}'$ being topologically closed. Since the latter two languages are also non-empty, it follows that $\mathcal{A} \cap \mathcal{L}' \neq \emptyset$. Inspecting Eq. (42), all operations retain regularity. □

Non-Falsifiable Assumptions and Strong Non-Anticipation

Proposition 8. Given an alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subsetneq \Sigma$, consider the instance of Problem 1, w.r.t. $U = \Sigma_{\mathrm{uc}}$, $Y = \Sigma - \Sigma_{\mathrm{uc}}$, $(\operatorname{pfx}\mathcal{A}, \mathcal{A})$ and $\mathcal{G}$. Furthermore let H be the game graph induced by $\mathcal{A}$ and $\mathcal{G}$ via M in (7) with acceptance condition $\{T^0, T^1\}$. Then player 0 has a winning strategy in the Büchi game (H, T^0) iff $\epsilon \in \operatorname{cfx}_{\operatorname{clo}\mathcal{A},Y}\mathcal{A}$.

Proof. $\Rightarrow$ Assume there exists a winning strategy f^0 for player 0 in the Büchi game (H, T^0). Then we know that all plays compliant with f^0 have corresponding words $\alpha \in \mathcal{A}$. Now use f^0 to define a map $r : Y^+ \to U$ s.t. $f^0(q) = r(\mathrm{p}_Y\, s)$ iff $\delta(q_0, s) = q$ (which is possible as f^0 always allows all outputs by definition). Then we can infer from Lemma 2 that the behaviour $\mathcal{L}$ associated with r fulfills (RM1)-(RM3) (with inverted inputs and outputs) and we have $\mathcal{L} \subseteq \mathcal{A}$ from the above construction. Now we can essentially apply the transformation from Section 4.2.2 (by swapping inputs and outputs) to obtain $\mathcal{L}' := \mathcal{L} \cup ((\mathrm{pfx}\,\mathcal{L})((\Sigma - \Sigma_{\mathrm{uc}})^\omega))$ satisfying (SC1) and (SC2) (with inverted controllable and uncontrollable events) and still having $\mathcal{L}' \subseteq \mathcal{A}$. Using $\mathcal{V} = \mathcal{L}'$, this implies that $\mathcal{V} \subseteq \mathcal{A}$, (i) $\mathcal{V} = \mathrm{clo}\,\mathcal{V}$ and (ii) $\mathrm{pfx}\,\mathcal{V}Y \cap \mathrm{pfx}\,\mathcal{A} \subseteq \mathrm{pfx}\,\mathcal{V}$. Using Definition 2 implies $\epsilon \in \mathrm{cfx}_{\mathrm{clo}\,\mathcal{A},\,Y}(\mathcal{A})$ and hence proves the statement.

$\Leftarrow$ Now assume that $\epsilon \in \mathrm{cfx}_{\mathrm{clo}\,\mathcal{A},\,Y}(\mathcal{A})$. Using Definition 2 and $Y = \Sigma - \Sigma_{uc}$ as uncontrollable events, this implies the existence of $\mathcal{V} \subseteq \mathcal{A}$ s.t. (i) $\mathcal{V} = \mathrm{clo}\,\mathcal{V}$ and (ii) $\mathrm{pfx}\,\mathcal{V}Y \cap \mathrm{pfx}\,\mathcal{A} \subseteq \mathrm{pfx}\,\mathcal{V}$. Now define $f : \Sigma^* \to \Xi$ with $\Xi := \{\, \xi \subseteq \Sigma \,|\, \Sigma - \Sigma_{\mathrm{uc}} \subseteq \xi \,\}$ s.t. $f(s) := \{\, \sigma \in \Sigma \,|\, s\sigma \in \mathrm{pfx}\,\mathcal{V} \,\} \cup \Sigma - \Sigma_{\mathrm{uc}}$. Now we can essentially apply the transformation from Section 4.2.2 (by swapping controllable and uncontrollable events) to obtain $\mathcal{V}''$ in (36) (with swapped inputs and outputs) which fulfils (RM1)-(RM3) (with inverted inputs and outputs) and $\mathcal{V}'' \subseteq \mathcal{A}$. Hence, we can infer from Lemma 1 that $\mathcal{V}''$ defines a map $r : Y^+ \to U$ s.t. $r(v)$ is the unique element of the singleton set $\{\, u \in u \mid \exists\, s \in (UY)^*\,.\ \mathrm{p}_Y\, s = v \wedge su \in \mathrm{pfx}\,\mathcal{V}'' \,\}$ for all $v \in Y^+$. Using r, define a player 0 strategy s.t. $f^0(q) = r(\mathrm{p}_Y\, s)$ iff $\delta(q_0, s) = q$. Now it remains to show that f^0 is a winning strategy in the Büchi game (H, T^0). To see this, consider a play π compliant with f^0 and its corresponding word α. Observe that (RM2) holding for $\mathcal{V}''$ and the construction of r implies that $\alpha \in \mathrm{pfx}\,\mathcal{V}''$. Then it follows from topological closeness (RM1) that $\alpha \in \mathcal{V}'' \subseteq \mathcal{A}$. As $\mathcal{A} = \mathrm{L}^\omega_{\mathrm{m}}(M_{\mathcal{A}})$, this implies that π visits T^0 infinitely often, what proves the statement. □

Proposition 9. Given an alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subsetneq \Sigma$, let $U = \Sigma_{\mathrm{uc}}$ and $Y = \Sigma - \Sigma_{\mathrm{uc}}$. For an assumption $\mathcal{A} \subseteq (UY)^\omega$ and a guarantee $\mathcal{G} \subseteq (UY)^\omega$, let $\mathcal{L}$ denote a solution to the reactive synthesis problem, Problem 1. If (44) holds then $\mathcal{A}$ and $\mathcal{L}$ are non-conflicting.

Proof. Given any $s \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L})$, we first refer to the absence of deadlocks, Eq. (4), and optionally extend s by one symbol to obtain $s \leq s' \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L}) \cap (UY)^*$. Referring to Eq. (44) and Definition 2, we can choose $\mathcal{V} \subseteq \mathcal{A} \cap (s'\Sigma^\omega)$ such that (i) $\mathcal{V}$ is relatively topologically closed w.r.t. $(\mathrm{clo}\,\mathcal{A}) \cap (s'\Sigma^\omega)$ and (ii) $((\mathrm{pfx}\,\mathcal{V})Y) \cap (\mathrm{pfx}\,\mathcal{A}) \cap (s'\Sigma^*) \subseteq (\mathrm{pfx}\,\mathcal{V})$. Since $(\mathrm{clo}\,\mathcal{A}) \cap (s'\Sigma^\omega)$ is topologically closed, (i) implies that $\mathcal{V}$ is closed, too. We will construct a strictly monotone sequence $(t_i)_{i\in\mathbb{N}}$ with $t_i < t_{i+1}$ and

$$s't_i \in (\mathrm{pfx}\,\mathcal{V}) \cap (\mathrm{pfx}\,\mathcal{L}) \cap (UY)^*\,. \tag{74}$$

We begin with $t_1 = \epsilon$ and assume, for some $i \in \mathbb{N}$, that we are provided a qualifying t_i. We can then pick $u \in U$ such that $s't_iu \in \mathrm{pfx}\,\mathcal{V}$ to observe $s't_iu \in \mathrm{pfx}\,\mathcal{L}$ by the free input (RM2). Likewise, we refer to the absence of deadlocks, Eq. (4), and choose $y \in Y$ such that $s't_iuy \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L})$. By property (ii) of $\mathcal{V}$, we obtain $s't_iuy \in (\mathrm{pfx}\,\mathcal{V}) \cap (\mathrm{pfx}\,\mathcal{L})$. Now $t_{i+1} := t_iuy$ satisfies the requirements and concludes the iterative construction of $(t_i)_{i\in\mathbb{N}}$. Denote β the singleton limit of the sequence. Then $s'\beta$ has infinitely many prefixes in $\mathrm{pfx}\,\mathcal{V}$ and $\mathrm{pfx}\,\mathcal{L}$. Topological closedness of $\mathcal{V}$ and $\mathcal{L}$ then implies that $s < s'\beta \in \mathcal{V} \cap \mathcal{L} \subseteq \mathcal{A} \cap \mathcal{L}$, and, hence, $s \in \mathrm{pfx}(\mathcal{A} \cap \mathcal{L})$. □

Proof of Theorem 5. As in the proof of Theorem 4, we refer to Proposition 7 to obtain (SC1), (SC2), $\mathcal{A} \cap \mathcal{L}' \subseteq \mathcal{G}$, $\mathcal{A} \cap \mathcal{L}' = \mathcal{A} \cap \mathcal{L}$, and $(\operatorname{pfx} \mathcal{A}) \cap (\operatorname{pfx} \mathcal{L}') = (\operatorname{pfx} \mathcal{A}) \cap (\operatorname{pfx} \mathcal{L})$. For non-conflictingness, pick $s \in (\operatorname{pfx} \mathcal{A}) \cap (\operatorname{pfx} \mathcal{L}')$, and refer to Proposition 9 for $s \in \operatorname{pfx}(\mathcal{A} \cap \mathcal{L}) = \operatorname{pfx}(\mathcal{A} \cap \mathcal{L}')$. Again, non-conflictingness and non-emptyness of $\mathcal{A}$ and $\mathcal{L}'$ imply a non-empty closed-loop behaviour. □