Discrete Event Dynamic Systems
https://doi.org/10.1007/s10626-019-00299-5

On the relation between reactive synthesis and supervisory control of non-terminating processes

Anne-Kathrin Schmuck[1] · **Thomas Moor**[2] · **Rupak Majumdar**[1]

Received: 8 December 2018 / Accepted: 7 November 2019 /

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 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.

Keywords Supervisory control · Reactive synthesis · ω-languages · Non-falsifiable assumptions

This article belongs to the Topical Collection: *Topical Collection on Theory-2020*
Guest Editors: Francesco Basile, Jan Komenda, and Christoforos Hadjicostis

✉ Anne-Kathrin Schmuck
akschmuck@mpi-sws.org

Extended author information available on the last page of the article.

1 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*.

The synthesis problem here is to construct a reactive module realised 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 by Büchi and Landweber (1969), and Rabin (1972). It is since then an active field of research, 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, 2007; Chatterjee and Henzinger 2012), and, most relevant for this paper, environment assumptions (see Bloem et al. 2014; Brenguier et al. 2017 for an overview). For a comprehensive introduction to the field see e.g. Thomas (1995) and 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 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; Cieslak et al. 1988; Cai et al. 2015; Yin and Lafortune 2016, 2017), robustness (e.g. Cury and Krogh 1999; Bourdon et al. 2005), modularity (e.g. Ramadge and Wonham 1989b; Rudie and Wonham 1992; 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, behaviours over an infinite time-axis (e.g. Ramadge 1989a; Thistle and Wonham 1994a, 2009; Moor et al. 2011; Baier and Moor 2015; Zhang and Cai 2018).

Both design methodologies seek to synthesise a causal feedback map that operates on a finite alphabet and satisfies a formal specification. When used on 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 uniformly 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 reasonably 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) (see Pnueli 1977), it is well understood how such formulae can be translated into ω-automata (see Vardi and Wolper 1986; Safra 1988).

Regarding *supervisory control*, most of the literature, including the seminal work by 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 1989a; Thistle and Wonham 1994a, 1995), where the authors explicitly relate their work to Church's problem. For a comprehensive introduction to SCT for ω-languages see also the technical report by 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 and provide an algorithm over deterministic Büchi-automata realisations which ensures that the resulting reactive module 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 (I) achieves reactive modules which do not falsify the assumption. The reverse transformation in (II), however, only holds if the computed reactive module returns solutions which are non-conflicting. We identify three sufficient conditions for the latter to hold: (a) topologically closed, (b) topologically closed input/output, and (c) strongly non-anticipating input/output plant (resp. environment) behaviours.

When establishing result (II)(c), we identify a close relationship 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)[1]) which ensure an almost identical form of non-conflictingness and which were derived independently in both communities.

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

Both synthesis algorithms are formalised as fixed-points in the μ-calculus. The RS algorithm uses a three-nested fixed-point while SCT synthesis amounts to a four-nested fixed-point. Result (I) clarifies that this additional fixed-point iteration indeed generates an additional property on the solution. Notably, this property cannot be encoded as an ω-regular property and hence, the SCT fixed-point 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 a 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 two-nested one. Using result (III)(b), the latter establishes equivalence of both algorithms in this case.

Related work The problem of correctly handling assumptions in synthesis has recently gained attention in the reactive synthesis community. As our solution via SCT synthesis and result (I) does not assume precise knowledge about the environment strategy (or the ability to impose an environment strategy), it is distinct from assume-guarantee approaches (Chatterjee and Henzinger 2007) or rational synthesis (Fisman et al. 2010). It is closest related to obliging games (Chatterjee et al. 2010), cooperative reactive synthesis as in Bloem et al. (2015) and assume-admissible synthesis (Brenguier et al. 2017). Chatterjee et al. (2010) incorporate a similar notion of non-conflictingness as SCT, but in contrast to SCT does not utilise liveness properties of the environment assumptions to enforce the system guarantees. Bloem et al. (2015) synthesise a reactive module which assumes as little cooperation (w.r.t. to a given hierarchy of cooperation levels) as possible by the environment. Contrary, our approach via result (I) always assumes the same form of cooperation coinciding with just one cooperation level. Applying assume-admissible synthesis, the system and its environment results in two individual synthesis problems. Given that both have a solution, only implementing the solution for the system ensures that the specification is fulfilled if the environment only takes *admissible* moves. This is comparable to the view taken in this paper, however, the hypothesis that the environment behaves *admissible* is stronger than the hypothesis used for SCT, namely that the environment attains its liveness properties if not prevented from doing so. Moreover, our approach only requires to solve one synthesis problem, instead of two.

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, 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.

Further, our work is in line with previous work establishing connections between supervisory control and other types of program synthesis, such as *behavior composition* (Barati and St-Denis 2015; Felli et al. 2017) or *planning by model checking* (Barveau et al. 1998). These works also discovered a strong connection between both fields which is used to apply established tools from one field to a problem statement of the other.

Outline This paper is organised as follows. After recalling necessary notation and basic facts in Section 2, we give a concise but self-contained introduction to reactive synthesis and supervisory control in Sections 3 and 4, respectively. Here, synthesis problems are stated to respect the conventions used in the respective literature, with an additional uniform characterisation of solutions in terms of ω-languages to facilitate the comparison in Section 5, in which we develop our main results (I)–(III) as outlined above. Technical propositions and proofs are organised in the Appendices A–D. A preliminary version of this work appeared as Schmuck et al. (2018), focusing on input-output behaviours. The current paper extends this by establishing a solution of a synthesis problem from SCT via RS for arbitrary topologically closed behaviours in Section 5.2. Additionally, we provide more in-depth explanations, examples and proofs for the results already presented by Schmuck et al. (2018).

2 Preliminaries

We provide common terminology and recall some elementary facts regarding formal languages, automata, two-player games and fixpoint calculus. A general introduction to these topics can be found in e.g. Hopcroft and Ullman (1979), Thomas (1990), Grädel et al. (2002), and Bradfield and Stirling (2006).

Formal languages Let Σ be a finite alphabet. Then we write Σ^*, Σ^+, and Σ^ω for the sets of finite sequences, non-empty finite sequences, and infinite sequences over Σ, respectively. 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 $\Psi \subseteq \Sigma$, the *natural projection* of $w \in \Sigma^*$ on Ψ^* is denoted by $\mathrm{p}_\Psi w$. As with all other operators on words used in this paper, we take pointwise images for an extension to languages over Σ, i.e., we write $\mathrm{p}_\Psi L$ for $\{\mathrm{p}_\Psi s \mid s \in L\}$ with $L \subseteq \Sigma^*$. For two words $s \in \Sigma^*$ and $t \in \Sigma^\infty$ we write $st \in \Sigma^\infty$ for the concatenation. We write $s \preceq t$ and $s \prec t$ if s is a prefix of t or a strict prefix of t, respectively. The set of all prefixes of a word $t \in \Sigma^\infty$ is denoted $\mathrm{pfx}t \subseteq \Sigma^*$. For $L \subseteq \Sigma^*$, we have $L \subseteq \mathrm{pfx}L$, and, if equality holds, we say that L is *prefix closed*. 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 $\mathrm{clo}\mathcal{L} := \lim\mathrm{pfx}\,\mathcal{L}$ as the *topological closure* of $\mathcal{L} \subseteq \Sigma^\omega$. $\mathcal{L}$ is said to be *topologically closed* if $\mathcal{L} = \mathrm{clo}\mathcal{L}$, and *relatively topologically closed w.r.t.* $\mathcal{M} \subseteq \Sigma^\omega$, if $\mathcal{L} = (\mathrm{clo}\mathcal{L}) \cap \mathcal{M}$. This notion of closedness 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.

Automata An *automaton* over the alphabet Σ is a tuple $M = (Q, \Sigma, \delta, Q_\mathrm{o})$ with the *state set* Q, the *transition relation* $\delta \subseteq Q \times \Sigma \times Q$ and the set of *initial states* $Q_\mathrm{o} \subseteq Q$. M is called finite if Q and δ are finite. We identify δ with its respective set-valued map $\delta : Q \times \Sigma \rightsquigarrow Q$ where $\delta(q, \sigma) := \{q' \mid (q, \sigma, q') \in \delta\} \subseteq Q$, and with the common inductive extension to a word-valued second argument $s \in \Sigma^*$. For $P \subseteq Q$ and $N \subseteq \Sigma^*$, we denote $\delta(P, N) := \cup\{\delta(q, s) \mid q \in P, s \in N\} \subseteq Q$ the image of $P \times N$ under δ. 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. We define $L = \mathrm{L}^*(M) := \{s \in \Sigma^* \mid \delta(Q_\mathrm{o}, s) \neq \emptyset\}$ and $\mathcal{L} = \mathrm{L}^\omega(M) := \{\alpha \in \Sigma^\omega \mid \mathrm{pfx}\,\alpha \subseteq \mathrm{L}^*(M)\}$ as the $*$- and ω-languages *generated* by M, respectively, which are prefix closed and topologically closed, respectively.

Accepted languages Given a set of *final states* $F \subseteq Q$ and the extended automaton tuple $M = (Q, \Sigma, \delta, Q_o, F)$, its *accepted* $*$-language is defined by $L_m^*(M) := \{s \in \Sigma^* \mid \delta(Q_o, s) \cap F \neq \emptyset\}$. *Regular* $*$*-languages* are those that are accepted by some finite automaton.

For ω-languages, we refer to an *acceptance condition* $\mathcal{F}$ and the extended automaton tuple $M = (Q, \Sigma, \delta, Q_o, \mathcal{F})$. A run π of M is an infinite sequence of states $q_1q_2q_3 \cdots \in Q^\omega$ and corresponds to the ω-word $\alpha = \sigma_1\sigma_2\sigma_3 \cdots \in \Sigma^\omega$ if $q_1 \in Q_o$ 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 $\mathrm{Inf}\pi$. We use *Büchi*, *generalized Büchi* and *Parity* acceptance conditions. The *Büchi acceptance condition* is given by a set $\mathcal{F} \subseteq Q$ and a *run* π over M is accepted if $(\mathrm{Inf}\pi) \cap \mathcal{F} \neq \emptyset$. Similarly, the *generalized Büchi acceptance condition* is given by a set $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$ s.t. $F_i \subseteq Q$, and a run π is accepted if $(\mathrm{Inf}\pi) \cap F_i \neq \emptyset$ for all $i \in \{1, \ldots, k\}$. Finally, the *parity acceptance condition* is given by a set of colors $\mathcal{F} = \{C_1, C_2, \ldots, C_k\}$ which is defined by a coloring function $c : Q \to \{1, ..., k\}$, s.t. $C_k = \{q \in Q\} c(q) = k$, and a run π is accepted if the highest color visited infinitely often is even, i.e., if $\max \mathrm{Inf} c(\pi)$ is even. An automaton M with Büchi, generalized Büchi or Parity acceptance condition is referred to as a Büchi, generalized Büchi or Parity automaton, respectively. The accepted ω-language $L_m^\omega(M)$ consists of all words $\alpha \in \Sigma^\omega$ for which there exists a corresponding accepted run over M. For deterministic automata, we have $L_m^\omega(M) = \lim L_m^*(M)$. 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.

Two-player games Let $\Sigma = \Sigma^0 \cup \Sigma^1$ be a disjoint composition of symbols and let $M = (Q^0 \cup Q^1, \Sigma^0 \cup \Sigma^1, \{q_0\}, \delta^0 \cup \delta^1, \mathcal{F})$ be a deterministic automaton s.t. $q_0 \in Q^0$, $\delta^0 \subseteq Q^0 \times \Sigma^0 \times Q^1$ and $\delta^1 \subseteq Q^1 \times \Sigma^1 \times Q^0$. Then the tuple $H = (Q^0, Q^1, \Sigma^0, \Sigma^1, \delta^0, \delta^1)$ defines the turn-based two player game graph induced by M, where Q^l, Σ^l and δ^l with $l \in \{0, 1\}$ are interpreted as the player l state set, alphabet and transition set, respectively. Any run π of M is called a play over H. Given a game graph H, a strategy for player 0 is a function $f^0 : (Q^0Q^1)^*Q^0 \to \Sigma^0$. A play π is *compliant* with f^0 if for all $k \in \mathbb{N}$ with $\pi(k) \in Q^0$ it holds that $\pi(k+1) = \delta^0(\pi(k), f^0(\pi|_{[1,k]}))$, where $\pi|_{[1,k]} = \pi(1)\pi(2)\ldots\pi(k)$. Strategies for player 1 are defined likewise, i.e., as functions $f^1 : (Q^0Q^1)^+ \to \Sigma^1$ and with a play π being compliant with f^1 if $\pi(k+1) = \delta^1(\pi(k), f^1(\pi|_{[1,k]}))$ for all $k \in \mathbb{N}$ with $\pi(k) \in Q^1$. A strategy $f^l, l \in \{0, 1\}$, is *memoryless* if $f^l(\nu q) = f^l(q)$ for all $\nu \in (Q^0 \cup Q^1)^*$ such that νq is in the domain of f^l. Throughout this paper, we discuss games from the perspective of exclusively one of the two players and interpret the Büchi (reps. parity) acceptance condition $\mathcal{F}$ for this player. The tuple $(H, \mathcal{F})$ if it is an accepted run in M. A strategy $f^l, l \in \{0, 1\}$, is a *winning strategy for player l* if all plays compliant with f^l are winning.

Fixpoint calculus We utilize the following notational conventions from the μ-calculus. Let f denote a monotone operator on a finite set Q, i.e., $f(P') \subseteq f(P'') \subseteq Q$ for all $P' \subseteq P'' \subseteq Q$. Then the least and the greatest fixed point exist uniquely and are denoted $\mu P.f(P)$ and $\nu P.f(P)$, respectively. They can be computed by the iterations $P_1 := \emptyset$, $P_{i+1} := P_i \cup f(P_i)$, with $\mu P.f(P) = \cup\{P_i \mid i \in \mathbb{N}\}$, and $P_1 := Q$, $P_{i+1} := P_i \cap f(P_i)$, with $\nu P.f(P) = \cap\{P_i \mid i \in \mathbb{N}\}$. If f is given as an expression in terms of multiple set-valued parameters with range Q, and if this expression is monotone in each parameter, so

are the respective fixed points. For such *monotone expressions* f, fixed-point formulae can be nested.

3 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.

3.1 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 :

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

If, in addition, r is implemented as a finite automaton, then $\mathcal{L}$ is ω-regular.[2] From the infinite time behaviour $\mathcal{L}$ we recover the *local behaviour* by taking the prefix pfx $\mathcal{L}$ and obtain a representation on a per-computation-cycle basis. Since the behaviour of a reactive module is exclusively characterised by individual computation cycles, we have $\mathcal{L} = \lim \mathrm{pfx}\ \mathcal{L} =: \mathrm{clo}\mathcal{L}$ and, hence, $\mathcal{L}$ is topologically closed; see also Lemma 1.

For our subsequent discussion we will eliminate the explicit reference to the reactive module $r : U^+ \to Y$ by utilising a direct characterisation of those languages $\mathcal{L}$ that qualify for a representation by Eq. 2. Referring to *behavioural systems theory* by J. C. Willems (1991), we adapt the notion of *input-output systems* to the notation used in the present paper.[3]

Definition 1 Given an ω-language $\mathcal{L} \subseteq (UY)^\omega$ or $\mathcal{L} \subseteq (YU)^\omega$ of alternating inputs and outputs, with $U, Y \neq \emptyset$ and $U \cap Y = \emptyset$, we say that

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

(ii) the *output locally processes the input* if
$\forall s \in \mathrm{pfx}\ \mathcal{L} \,.\, \forall\, y', y'' \in Y \,.\, sy' \in \mathrm{pfx}\ \mathcal{L} \wedge sy'' \in \mathrm{pfx}\ \mathcal{L} \ \to\ y' = y''$.

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

[3] As Willems (1991) addresses the time axis $\mathbb{R}$ and $\mathbb{Z}$, there is no exact technical correspondence. Still, for topologically closed languages the intention of a *locally free input* matches Def. VIII.1 and VIII.4 and the intention of an *output to locally process the input* matches Definition VIII.3 in Willems (1991).

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

Lemma 1 *Let $U, Y \neq \emptyset$, $U \cap Y = \emptyset$ and $r : U^+ \to Y$. Then the associated behaviour $\mathcal{L} \subseteq (UY)^\omega$ of r defined by* Eq. 2 *is non-empty and it holds that:*

(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

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

for $v \in U^+$.

3.2 Problem statement (RS)

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 $\mathcal{G}$ may express liveness properties which can not be characterised on a per-computation-cycle basis. Hence, we do not necessarily have equality in $\mathcal{G} \subseteq \lim \mathrm{pfx}\, \mathcal{G}$; i.e., $\mathcal{G}$, in contrast to $\mathcal{L}$, may not 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 parameterise such assumptions by an ω-language $\mathcal{A} \subseteq (UY)^\omega$ to express that over infinitely many computation cycles the environment will produce input symbols such that we end up with an ω-word $\alpha \in \mathcal{A}$. The intention here is to synthesise the system according to the inclusion $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G} := ((U \cup Y)^\omega - \mathcal{A}) \cup \mathcal{G}$ and to then conclude $\alpha \in \mathcal{G}$ for any $\alpha \in \mathcal{A}$. A consequence of this concept of an environment assumption is that over finitely many computation cycles the environment must comply to the local behaviour $\mathrm{pfx}\, \mathcal{A}$. This in turn restricts the output symbols that the reactive module may generate in each individual computation cycle in order be operational over infinitely many cycles. Technically, we require that *the system and the environment do not deadlock*, i.e.,

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

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.[4]

Note 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 $\emptyset \neq \mathcal{G} \subseteq \mathcal{A} \subseteq (UY)^\omega$. Furthermore, 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.

Remark 1 The problem of reactive synthesis is more commonly formalised 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, determinising this automaton to obtain a deterministic Rabin or Parity automaton.

This transformation is well understood (Vardi and Wolper 1986; Safra 1988); see e.g. Finkbeiner (2016) for a comprehensive discussion.

Remark 2 In the reactive synthesis literature, environment assumptions are usually specified by a separate LTL formula which is then translated to an automaton as outlined in Remark 1, or they are directly given as an automaton model. In either case, the automaton may exhibit reachable states from which on the acceptance condition can not be satisfied, i.e., we may have that the generated language $A_{\text{loc}} \subseteq (UY)^*$ is a strict super-set of the prefix pfx $\mathcal{A}$ of the accepted language $\mathcal{A}$. The requirement of the system and the environment not to deadlock, Eq. 4, is then substituted by

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

However, if we synthesise a system with behaviour $\mathcal{L}$ in which there indeed exist words $s \in A_{\text{loc}} \cap (\text{pfx}\ \mathcal{L})$ with $s \notin \text{pfx}\ \mathcal{A}$, any extension $\alpha \in (UY)^\omega$, $s < \alpha$, of such a word is not in $\mathcal{A}$. Hence, $\mathcal{L}$ *falsifies the assumption* and we may after infinitely many computation cycles end up with an ω-word $\alpha \notin \mathcal{G}$. This is undesirable and we will identify further more subtle variations of this issue in due course of our study. To this end, we restrict the discussion to the case of $A_{\text{loc}} = \text{pfx}\ \mathcal{A}$, i.e., we parameterise the assumption as a single ω-language $\mathcal{A}$ and we refer to its prefix pfx $\mathcal{A}$ as the associated local behaviour.

3.3 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 input $u \in U$ and

[4]For $\mathcal{A} = \emptyset$ the upper bound $\mathcal{A} \to \mathcal{G}$ degenerates and the specification becomes $\mathcal{L} \subseteq (UY)^\omega$. Whenever convenient, we therefore assume $\mathcal{A} \neq \emptyset$ and, likewise, $\mathcal{G} \neq (UY)^\omega$.

the system selects the output $y \in Y$. It was shown by Gurevich and Harrington (1982) and 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. While this does not imply that $\mathcal{A} \to \mathcal{G}$ can be realised by a deterministic Büchi automaton, there still exists 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 generalised Büchi automaton M with acceptance condition $\mathcal{F} = \{T^0, T^1\}$ s.t. $\mathcal{A} = \mathrm{L}_{\mathrm{m}}^{\omega}(M_{\mathcal{A}})$, $\mathcal{G} = \mathrm{L}_{\mathrm{m}}^{\omega}(M_{\mathcal{G}})$ 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

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

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

The generalised Büchi automaton M defines the turn-based deterministic game graph $H = (Q^0, Q^1, U, Y, \delta^0, \delta^1)$. 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 $\mathcal{C} = \{\emptyset, \setminus QT^0, T^0, T^1\}$, where $C_2 = \setminus QT^0$ 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

$$\mathsf{Win}^1 = \nu X_4.\mu X_3.\nu X_2.\mu X_1.\textstyle\bigcup_{k=1}^{4}(C_k \cap \mathsf{Pre}^1(X_k)), \tag{8}$$

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

$$\mathsf{Pre}^1(A) = \{q^0 \in Q^0 \mid \forall u \in U \,.\, \delta^0(q^0, u) \in A\} \cup \{q^1 \in Q^1 \exists\, y \in Y \,.\, \delta^1(q^1, y) \in A\}. \tag{9}$$

However, as C_1 is empty, Eq. 8 simplifies to the three-nested fixed point

$$\mathsf{Win}^1 = \nu X_4 \,.\, \mu X_3 \,.\, \nu X_2 \,.\, (T^1 \cap \mathsf{Pre}^1(X_4)) \cup \mathsf{Pre}^1(X_3) \cup ((Q \setminus T^0) \cap \mathsf{Pre}^1(X_2)). \tag{10}$$

If $q_0 \in \mathsf{Win}^1$, the synthesis problem has a solution and a memoryless winning strategy for the system player can be derived from the iterations in Eq. 10 as follows.

[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.

Consider the last iteration of the fixed-point in Eq. 10 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 $X_4^\infty = \bigcup_{i=0}^{k} X_3^i$ 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.

$$\mathsf{rank}(q) = i \quad \text{iff} \quad q \in X_3^i \setminus X_3^{i-1} \quad \text{for} \quad 1 \leq i \leq k. \tag{11}$$

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 $y = f^1$ 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 Eq. 1 in the obvious way s.t. $r(\mathrm{p}_U s) = f^1(q)$ iff $\delta(q_0, s) = q$. A finite automaton realising 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. $y \not/ f^1(q)$. A proof that this winning strategy indeed defines a reactive module solving Problem 1 can be obtained as a special case of the construction presented in Bloem et al. (2012).

Remark 3 Referring back to Remark 2, the outlined synthesis algorithm generalises to the case where $\mathcal{G}$ is not necessarily a subset of $\mathcal{A}$ and $\mathrm{pfx}\,\mathcal{A} \subsetneq A_{\mathrm{loc}}$. 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 which generates $A_{\mathrm{loc}} = \mathrm{L}^*(M)$.

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 Eq. 10 simplifies to

$$\mathsf{Win}^1 = \nu X_4 \,.\, \mu X_3 \,.\, \mathsf{Pre}^1(X_3) \cup (T^1 \cap \mathsf{Pre}^1(X_4)) \,. \tag{12}$$

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 Eq. 12; see e.g. Maler et al. (1995) and Zielonka (1998). In this context, the basic version of reactive synthesis without environment assumptions results in computing 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 realisable 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. 12 over M in Eq. 7 instead of M^1.

Example 1 Consider a topologically closed assumption $\mathcal{A}$ as discussed in Remark 4 s.t. $\mathrm{pfx}\,\mathcal{A}$ is the language generated by M depicted in Fig. 1 and $\mathcal{G}$ is accepted by M with final

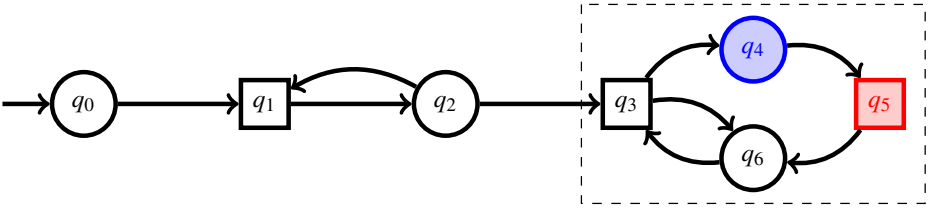

Fig. 1 Transition structure of the automaton M in Eq. 7 discussed in Example 1-2. Environment and system states Q^0 and Q^1 are indicated by circles and squares, respectively. The final states $T^0 = \{q_4\}$ and $T^1 = \{q_5\}$ are indicated in blue and red, respectively

state set $T^1 = \{q_5\}$. Using Eq. 12 to solve this synthesis problem results in the winning state set $\mathsf{Win}^1 = \{q_3, q_4, q_5, q_6\}$ (indicated by the dashed square in Fig. 1).

As $q_0 \notin \mathsf{Win}^1(F^1)$, Problem 1 has no solution in this example. If q_3 would be an initial state, it should be noted that the resulting system strategy would transition from q_3 in q_4 as $\mathsf{rank}(q_3) = 3$, $\mathsf{rank}(q_4) = 2$ and $\mathsf{rank}(q_6) = 4$.

Example 2 Consider the assumption $\mathcal{A}$ given by the language accepted by M in Fig. 1 with accepting state set $T^0 = \{q_4\}$. In this case, we evaluate (10) and obtain $\mathsf{Win}^1 = Q$. Using Eq. 11, all states in $Q \setminus T^0$ have rank 1 while q_4 has rank 2. This implies that the resulting system strategy will transition from q_3 to q_6.

The problem discussed in Example 1 fails to have a solution, as the environment can prevent the system from reaching q_3 from the initial state. 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. Furthermore, choosing to transition to q_4 or q_6 from q_3 is now 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. Example 2 shows that the constructed strategy actually favours a transition from q_3 to q_6, and by this falsifies the assumption.

4 Supervisory control

The purpose of this section is to give a concise introduction to supervisory control 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 (1989a) and Thistle and Wonham (1994a) 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 the synthesis problem for the specific case of deterministic Büchi automata realisations of the involved ω-languages.

4.1 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

$$f : \Sigma^* \to \Gamma\,, \quad \text{with} \quad \Gamma := \{\, \gamma \subseteq \Sigma \mid \Sigma_{\mathrm{uc}} \subseteq \gamma \,\}\,, \tag{13}$$

where $\Sigma_{\mathrm{uc}} \subseteq \Sigma$ 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.

For the subsequent discussion, the following representation of a supervisor as an ω-language turns out convenient: [6]

$$\mathcal{L} := \{\, \alpha \in \Sigma^\omega \mid \forall\, s \in \Sigma^* \,.\, \forall\, \sigma \in \Sigma \,.\, s\sigma < \alpha \;\rightarrow\; \sigma \in f(s) \,\}\,. \tag{14}$$

The language defined by Eq. 14 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 *Let* Σ *and* $\emptyset \neq \Sigma_{\mathrm{uc}} \subseteq \Sigma$. *Then the behaviour* $\mathcal{L} \subseteq \Sigma^\omega$ *in* Eq. 14 *associated with* $f : \Sigma^* \to \Gamma$ *is non-empty and exhibits the following properties:*

(SC1) $\mathcal{L}$ *is topologically closed, and*
(SC2) $\mathcal{L}$ *is universally controllable, i.e.,* $(\mathrm{pfx}\,\mathcal{L})\Sigma_{\mathrm{uc}} \subseteq \mathrm{pfx}\,\mathcal{L}$.

Vice versa, if $\emptyset \neq \mathcal{L} \subseteq \Sigma^\omega$ *satisfies (SC1) and (SC2), then* $f' : \Sigma^* \to \Gamma$ *defined by*

$$f'(s) := \{\, \sigma \in \Sigma \mid s\sigma \in \mathrm{pfx}\,\mathcal{L} \,\} \cup \Sigma_{\mathrm{uc}} \tag{15}$$

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

4.2 Problem statement (SCT)

The problem commonly referred to as *supervisory controller synthesis* is about the systematic design of a supervisor for a given plant, such that the resulting closed-loop system – established by the feedback composition of this supervisor with the plant – satisfies a given specification. When the supervisor and plant behaviours are given as the ω-languages $\mathcal{L} \subseteq \Sigma^\omega$ and $\mathcal{A} \subseteq \Sigma^\omega$, respectively, their closed-loop configuration evolves on words that comply with both component behaviours. Technically, we distinguish the *local closed-loop behaviour* $K_{\mathrm{loc}} := (\mathrm{pfx}\,\mathcal{L}) \cap (\mathrm{pfx}\,\mathcal{A})$ and the *accepted closed-loop behaviour* $\mathcal{K} := \mathcal{L} \cap \mathcal{A}$ and require that the latter meets an upper-bound specification $\mathcal{K} \subseteq \mathcal{G}$.

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. To rule out such sequences, the synthesis problem asks for a *non-blocking supervisor*, i.e., it is required that $\mathcal{L}$ and $\mathcal{A}$ are *non-conflicting*:

$$(\mathrm{pfx}\,\mathcal{L}) \cap (\mathrm{pfx}\,\mathcal{A}) = \mathrm{pfx}(\mathcal{L} \cap \mathcal{A})\,. \tag{16}$$

As the local behaviour K_{loc} of a non-conflicting closed loop can be recovered by $K_{\mathrm{loc}} = \mathrm{pfx}\,\mathcal{K}$, one refers 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. 16, is equivalent to the *absence of deadlocks*, Eq. 4.

The interpretation of the case when $\mathcal{A}$ is not topologically closed is more involved. Here, Eq. 16 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

[6] 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.

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 5.3.

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

Problem 2 (Supervisory Controller Synthesis) Given an alphabet Σ with *uncontrollable event set* $\emptyset \neq \Sigma_{\mathrm{uc}} \subseteq \Sigma$, a *plant* $\mathcal{A} \subseteq \Sigma^\omega$ and an *upper-bound specification* $\mathcal{G} \subseteq \Sigma^\omega$, the *supervisory control problem* $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ asks to either construct a non-blocking supervisor with associated behaviour $\mathcal{L} \subseteq \Sigma^\omega$, see Eq. 16, such that

$$\emptyset \neq \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{G}\,, \tag{17}$$

or, to verify that no such supervisor exists[7].

Referring to the behavioural characterisation of supervisors, Lemma 2, we identify a qualifying associated behaviour $\mathcal{L}$ as a solution to the supervisory control problem. As in the setting of reactive synthesis, Section 3.2, 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 $A_{\mathrm{loc}} \subseteq \Sigma^*$ and $\mathcal{A} \subseteq \Sigma^\omega$, where A_{loc} is prefix-closed and represents the local behaviour. In this regard, Ramadge (1989a) and Thistle and Wonham (1994a) specifically address the case $\mathrm{pfx}\,\mathcal{A} \subsetneq A_{\mathrm{loc}}$ 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 $A_{\mathrm{loc}} \cup ((A_{\mathrm{loc}} - \mathrm{pfx}\,\mathcal{A})\dagger^*)$ and $\mathcal{A}$ by $\mathcal{A} \cup ((A_{\mathrm{loc}} - \mathrm{pfx}\,\mathcal{A})\dagger^\omega)$, to formally obtain $A_{\mathrm{loc}} = \mathrm{pfx}\,\mathcal{A}$. 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 $A_{\mathrm{loc}} = \mathrm{pfx}\,\mathcal{A}$ is not restrictive.

4.3 Algorithmic solution

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].

The following proposition from Ramadge (1989a) characterises this set.

Proposition 1 *Given an alphabet Σ with uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, 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*

[7] For $\mathcal{A} = \emptyset$ the problem trivially has no solution and for $\Sigma_{\mathrm{uc}} = \Sigma$ we have $\mathcal{L} = \Sigma^\omega$ for the only qualifying supervisor; i.e., in this case the synthesis problem collapses to the verification of $\mathcal{A} \subseteq \mathcal{G}$. Whenever convenient we therefore assume $\mathcal{A} \neq \emptyset$ and $\Sigma_{\mathrm{uc}} \neq \Sigma$.

[8] The style of argument is similar to the *Youla-Kučera parameterization* of all stabilising controllers for a linear time invariant system, which is conceived a 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 to care about the quantitative upper bound specification at an largely independent second stage.

(i) $\mathcal{K}$ *is relatively topologically closed w.r.t.* $\mathcal{A}$*, i.e.,* $\mathcal{K} = clo(\mathcal{K}) \cap \mathcal{A}$*, and*
(ii) $\mathcal{K}$ *is* $*$*-controllable w.r.t.* $\mathcal{A}$*, i.e.,* $((\text{pfx}\,\mathcal{K})\Sigma_{\text{uc}}) \cap (\text{pfx}\,\mathcal{A}) \subseteq (\text{pfx}\,\mathcal{K})$.

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

$$f(s) := \{\sigma \in \Sigma \mid s\sigma \in \text{pfx}\,\mathcal{K}\} \cup \Sigma_{\text{uc}} \tag{18}$$

for all $s \in \Sigma^*$.

Proposition 1 reduces Problem 2 to the synthesis of a closed-loop behaviour $\emptyset \neq \mathcal{K} \subseteq \mathcal{A} \cap \mathcal{G}$ satisfying (i) and (ii). Such a closed-loop behaviour exists, if and only if

$$\mathcal{K}^{\uparrow} := \cup\{\mathcal{K} \subseteq \mathcal{A} \cap \mathcal{G} \mid \mathcal{K} \text{ satisfies (i) and (ii) from Proposition 1}\} \tag{19}$$

is non-empty. Moreover, if $\mathcal{K}^{\uparrow}$ itself exhibits conditions (i) and (ii) then $\mathcal{K}^{\uparrow}$ is referred to as the *supremal closed-loop behaviour* and the so called *maximally permissive supervisor* to solve the synthesis problem can be extracted from $\mathcal{K}^{\uparrow}$ via (18). However, the union in Eq. 19 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, where maximally permissive strategies do not exist in general.

Remark 6 Condition (i) in Proposition 1 is specific for the supervision of ω-languages, whereas condition (ii) is literally identical to the original notion of controllability introduced by Ramadge and Wonham (1987) for the more common setting 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 (1989a). 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), focusing on the supremal closed-loop behaviour and a corresponding maximally permissive supervisor.

To compute some $\mathcal{K} \subseteq \mathcal{A} \cap \mathcal{G}$ which solves Problem 2, Thistle and Wonham (1994a) introduce the following notion of the controllability prefix, which characterises the set of states from which the synthesis problem has a solution.

Definition 2 Given an alphabet Σ with uncontrollable events $\Sigma_{\text{uc}} \subseteq \Sigma$ 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[9] $\text{cfx}_{\mathcal{A}}\mathcal{G}$ and defined as the set of strings $s \in \text{pfx}\,\mathcal{G}$, for which there exists $\emptyset \neq \mathcal{V} \subseteq \mathcal{A} \cap \mathcal{G} \cap (s\Sigma^{\omega})$ such that $\mathcal{V}$ is

(i) rel. topologically closed w.r.t. $\mathcal{A} \cap (s\Sigma^{\omega})$, i.e., $\mathcal{V} = clo(\mathcal{V}) \cap (\mathcal{A} \cap (s\Sigma^{\omega}))$, and
(ii) $*$-*controllable* w.r.t. $\mathcal{A} \cap (s\Sigma^{\omega})$, i.e., $((\text{pfx}\,\mathcal{V})\Sigma_{\text{uc}}) \cap (\text{pfx}\,\mathcal{A}) \cap (s\Sigma^*) \subseteq \text{pfx}\,\mathcal{V}$.

[9] When the role of uncontrollable events is not clear from the context, we write $\text{cfx}_{\mathcal{A},\,\Sigma_{\text{uc}}}\mathcal{G}$.

Comparing (i) and (ii) in Definition 2 with their analogue in Proposition 1, we see that $\mathcal{V}$ is a closed-loop behaviour that can be enforced by a non-blocking supervisor that "takes over to control the plant" after the string s in the controllability prefix $\mathrm{fx}_{\mathcal{A}}\mathcal{G}$ was generated by the plant. As $\mathcal{V} \subseteq \mathcal{G}$, this supervisor is able to enforce the guarantee $\mathcal{G}$.

It is shown in Thistle and Wonham (1994a) that $\epsilon \in \mathrm{cfx}_{\mathcal{A}}\mathcal{G}$ if and only if $\mathcal{K}^{\uparrow} \neq \emptyset$, i.e., if and only if Problem 2 has a solution.

For ω-regular parameters, the controllability prefix can be represented in terms of a fixed-point over an automaton representation of the involved languages; see Thistle and Wonham (1994a). For a concise representation of this construction, we assume again that both $\mathcal{G}$ and $\mathcal{A}$ are realisable as deterministic Büchi automata. We further assume without loss of generality that $\emptyset \neq \mathcal{G} \subseteq \mathcal{A}$. Given this setting, we consider a generalised Büchi automaton [10]

$$M = (Q, \Sigma, \{q_0\}, \delta, \{F_{\mathcal{A}}, F_{\mathcal{G}}\}) \tag{20}$$

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

Given the generalised deterministic Büchi automaton M, a string $s \in \mathrm{cfx}_{\mathcal{A}}\mathcal{G}$ 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. Following Thistle and Wonham (1994b) and Moor (2017), the set of all winning states can be computed by the four-nested fix-point

$$\mathsf{Win}(M) := \nu Z \,.\, \mu Y \,.\, \nu X \,.\, \mu W \,.\, \mathsf{Pre}((W \setminus F_{\mathcal{A}}) \cup Y \cup (F_{\mathcal{G}} \cap Z), X \setminus F_{\mathcal{A}}) \,, \tag{21}$$

where

$$\mathsf{Pre}(T,\ D) := \{\, q \in Q \mid \delta(q, \Sigma) \cap T \neq \emptyset \ \text{ and } \ \delta(q, \Sigma_{\mathrm{uc}}) \subseteq T \cup D \,\} \,, \tag{22}$$

denotes the *inverse dynamics operator*; i.e., $\mathsf{Pre}(T,\ D)$ denotes the set of predecessor states $q \in Q$ of the *target set* $T \subseteq Q$ which can be controlled such that the successor state violates the *domain constraint* $D \subseteq Q$ only in favour of attaining the target set.

Recall that a solution to Problem 2 exists if and only if $\epsilon \in \mathrm{cfx}_{\mathcal{A}}\mathcal{G}$, which amounts to $q_0 \in \mathsf{Win}(M)$. If this is true, a non-blocking supervisor solving Problem 2 can be derived from iterations of the fixed-point in Eq. 21 as follows. Consider the last iteration of the fixed-point in Eq. 21 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 $Z^{\infty} = \bigcup_{i=0}^{k} Y^i$ 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 $Y^i = X^i = \bigcup_{j=0}^{l} W_j^i$ 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.

$$\mathsf{rank}(q) = \begin{cases} (i, j) q \in (Y^i \setminus Y^{i-1}) \cap (W_j^i \setminus W_{j-1}^i),\ i, j > 0 \\ (0, 0) q \in Z^{\infty} \cap F_{\mathcal{G}} \end{cases} \tag{23}$$

[10] If $\mathcal{A}$ and $\mathcal{G}$ are represented by trim deterministic Büchi automata M^0 and M^1, M can be constructed in the same way as outlined in the context of reactive synthesis; see Footnote 5.

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

$$g(q) := \begin{cases} \Sigma_{\text{uc}} \cup \{\, \sigma \in \Sigma_{\text{c}} \mid \delta(q,\sigma)! \text{ and } \mathsf{rank}(\delta(q,\sigma)) < \mathsf{rank}(q) \,\} & \text{if } \mathsf{rank}(q) > (0,0) \\ \Sigma_{\text{uc}} \cup \{\, \sigma \in \Sigma_{\text{c}} \mid \delta(q,\sigma) \in Z^\infty \,\} & \text{otherwise.} \end{cases} \tag{24}$$

The state feedback map g defines a supervisor $f : \Sigma^* \to \Gamma$ in the obvious way, i.e. $f(s) = g(q)$ if $\delta(q_0, s) = q \in Z^\infty$ and $f(s) = \Sigma \in \Gamma$, otherwise. The behaviour $\mathcal{L}$ associated with f is defined via (14). The proof that $\mathcal{L}$ solves Problem 2 can be obtained as a special case of the construction presented by Thistle and Wonham (1992, 1994b).

Remark 7 The state feedback g, as defined in Eq. 24, 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 $\text{fx}_\mathcal{A}\mathcal{G}$ and only eventually to activate the supervisor constructed above. The original literature (Thistle and Wonham 1994a) 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 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 (1994a) 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. In this case the fixed-point in Eq. 21 collapses to

$$\begin{aligned} \mathsf{Win}(M) &= \nu Z \,.\, \mu Y \,.\, \mathsf{Pre}(Y \cup (F_\mathcal{G} \cap Z)), \\ &= \nu Z \,.\, \mu Y \,.\, \mathsf{Pre}(Y) \cup (F_\mathcal{G} \cap \mathsf{Pre}(Z)), \end{aligned} \tag{25}$$

where we use the short form $\mathsf{Pre}(T) := \mathsf{Pre}(T, \emptyset)$.

We see that Eqs. 12 and 25 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 $\mathcal{A}$, which is verified in Section 5.

Example 3 Consider the automaton M depicted in Fig. 2 and two sets of problem parameters: (A) $F_\mathcal{A} = F_\mathcal{G} = \{s\}$, $\Sigma_{\text{uc}} = \Sigma \setminus \{b\}$, and (B) $F_\mathcal{A} = \{r, s\}$, $F_\mathcal{G} = \{s\}$, $\Sigma_{\text{uc}} = \Sigma \setminus \{b\}$. In both cases the computation of Eq. 21 yields $\mathsf{Win}(M) = Q$ and $\mathsf{rank}(s) = (0,0)$,

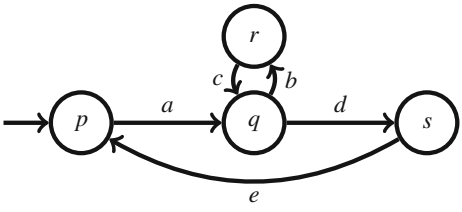

Fig. 2 Transition structure of automaton M in Eq. 20 representing $\mathcal{A}$ and $\mathcal{G}$ for the instance of Problem 2 which is solved in Example 3

$\mathsf{rank}(q) = (1, 1)$, and $\mathsf{rank}(r) = \mathsf{rank}(p) = (1, 2)$. Hence, the resulting supervisor disables b in q. However, given the corresponding problem instance we see, that in case (B) disabling b in q is strictly necessary, while this is not true for (A). In fact, as $\mathcal{G}$ and $\mathcal{A}$ coincide for case (A) there exists a maximally permissive supervisor (see the discussion in Remark 6) which enables both b and d in s. This also constitutes a solution to the given supervisory control problem, as we assume that the plant only generates runs which correspond to words in $\mathcal{A}$. By this we know that it will always eventually transition from q to s, implying that the resulting closed loop behaviour fulfils the guarantee.

Considering a third case (C) with problem parameters $F_{\mathcal{A}} = \{r, s\}$, $F_{\mathcal{G}} = \{s\}$, $\Sigma_{\text{uc}} = \Sigma$ the fixed-point computation amounts to $\mathsf{Win}(M) = \emptyset$, hence, the corresponding supervisory control problem has no solution. This is due to the fact that the controller cannot prevent the plant from alternating between r and q (as $b \in \Sigma_{\text{uc}}$) and by this generating an accepting run on $M_{\mathcal{A}}$ which is not in $\mathcal{G}$.

5 Comparison

This section provides a comparison between the reactive synthesis problem, Problem 1, as introduced in Section 3, and the supervisory control problem, Problem 2, as introduced in Section 4. For both problems, the system that one seeks to synthesise 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 synthesis 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.

In Section 5.1, 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. Furthermore, this solution results in a reactive module whose associated behaviour does not conflict with $\mathcal{A}$, i.e., the computed module does not falsify the assumptions. While this is formalised on the language level, we show though examples how this practically results in the algorithmic solution of a reactive synthesis problem via the four-nested fixed-point from supervisory control, s.t. the latter property is ensured.

The converse transformation, i.e., to solve a supervisory control problem by reactive synthesis, requires additional assumptions to ensure that the resulting supervisor is *non-conflicting*. Section 5.2 first discusses a general transformation which ensures the latter property if the plant behaviour $\mathcal{A}$ is topologically closed. This transformation is then simplified for the special case of plant behaviours with alternating controllable and uncontrollable events. Section 5.3 finally shows, that for the latter problem class a weaker condition,

namely strongly non-anticipation, exists, which ensures that the suggested transformation results in a supervisor which is *non-conflicting*. Again, all transformations are formalised on the language level and we show the resulting algorithmic solution procedures through examples.

5.1 Reactive synthesis via supervisory control

In this section, we show how a reactive synthesis problem can be solved using supervisory controller synthesis.

This is done in two steps. Given a particular instance of the reactive synthesis problem we (i) derive a corresponding supervisory control problem which we solve as outlined in Section 4.3, and (ii) convert the solution to a reactive module that solves the original reactive synthesis problem.

Step (i) 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 $\mathsf{SCT}[\Sigma,\ \Sigma_{\mathrm{uc}}, \mathcal{A},\ \mathcal{G}]$. To this end, we assume that $\mathsf{SCT}[\Sigma,\ \Sigma_{\mathrm{uc}}, \mathcal{A},\ \mathcal{G}]$ exhibits a solution $f : \Sigma^* \to \Gamma$ with associated behaviour $\mathcal{L}$.

Step (ii) As our first observation, we recall from Lemma 2 that the behaviour $\mathcal{L}$ 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

$$h(s) := \Sigma_{\mathrm{uc}} \cup \{\, \sigma \in f(s) \mid s\sigma \in \mathrm{pfx}\, \mathcal{A} \,\} \tag{26}$$

for all $s \in \Sigma^*$. This is not expected to affect the closed-loop behaviour and, indeed, the supervisor f constructed in Section 4.3 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

$$f'(s) = \Sigma_{\mathrm{uc}} \,\dot\cup\, \{\sigma\}\,,\ \text{where} \sigma \in \Sigma - \Sigma_{\mathrm{uc}} \text{ and, if } h(s) \neq \Sigma_{\mathrm{uc}}, \text{ then } \sigma \in h(s)\,, \tag{27}$$

for all $s \in \Sigma^*$.

As an example, f' can be constructed as a composition $f' = h' \circ h$ where $h' : \Gamma \to \Gamma$ is a static filter such that

$$h'(\gamma) = \Sigma_{\mathrm{uc}} \,\dot\cup\, \{\sigma\}\,,\ \text{where } \sigma \in \Sigma - \Sigma_{\mathrm{uc}} \text{ and, if } \gamma \neq \Sigma_{\mathrm{uc}}, \text{ then } \sigma \in \gamma\,, \tag{28}$$

for all $\gamma \in \Gamma$.

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:

$$\forall s \in \text{pre}\mathcal{L}' \,.\, \exists \sigma \in \Sigma - \Sigma_{\text{uc}} \,.\, s\sigma \in \text{pre}\mathcal{L}' \,, \tag{29}$$

$$\forall s \in \text{pre}\mathcal{L}' \,.\, \forall \sigma', \sigma'' \in \Sigma - \Sigma_{\text{uc}} \,.\, s\sigma' \in \text{pre}\mathcal{L}' \wedge s\sigma'' \in \text{pre}\mathcal{L}' \;\rightarrow\; \sigma' = \sigma'' \,, \tag{30}$$

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.,

$$\mathcal{L}'' := \mathcal{L}' \cap (UY)^\omega \,. \tag{31}$$

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 $\mathcal{A} \subseteq (UY)^\omega$.

Result We can now state our first main result, i.e., $\mathcal{L}''$ indeed solves the reactive synthesis problem $\mathsf{RS}[U,\ Y, \mathcal{A},\ \mathcal{G}]$. A proof is given in Appendix B.

Theorem 1 *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 $\Sigma_{\text{uc}} := U$. If a supervisor $f : \Sigma^* \rightarrow \Gamma$ with associated behaviour $\mathcal{L}$ solves the supervisory control problem* $\mathsf{SCT}[\Sigma,\ \Sigma_{\text{uc}}, \mathcal{A},\ \mathcal{G}]$, *then $\mathcal{L}''$, as defined by* Eqs. 26, 27 and 31, *solves* $\mathsf{RS}[U,\ Y, \mathcal{A},\ \mathcal{G}]$. *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 Theorem 1, for any reactive synthesis problem 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, we therefore achieve the solution of an RS problem via SCT by using the synthesis algorithm from supervisory control, Section 4.3, and the additional post-processing given by Eqs. 26-31, as demonstrated by Example 4. However, since the requirement of non-conflictingness in the context of SCT is stronger than the requirement of the absence of deadlocks in the context of RS, we may encounter the situation of a solvable RS problem, for which the corresponding SCT problem exhibits no solution at all; see the following remark for more detail.

Remark 9 Although $\mathcal{A}$ and $\mathcal{L}$ in Theorem 1 are non-conflicting by hypothesis, this property is in general not preserved by the proposed transformation, i.e., $\mathcal{A}$ and $\mathcal{L}''$ may fail to be non-conflicting. However, if all problem parameters are ω-regular and if a realisation of $\mathcal{L}$ is obtained by the synthesis algorithm presented in Section 4.3, the situation becomes more favourable. Here, the transformations in Eqs. 26–31 effectively restrict the state feedback $g\colon Q \rightarrow \Gamma$ given in Eq. 24 to enable exactly one controllable event in each state $q \in Q$ in which the plant can execute a controllable event at all. This restriction preserves the fact that controllable events are only enabled if the respective transition decreases the state rank or if the state rank in the current state is (0,0). Non-conflictingness of $\mathcal{A}$ and $\mathcal{L}''$ then follows exactly by the same arguments as for $\mathcal{A}$ and $\mathcal{L}$ provided by the original literature (Thistle and Wonham 1992, 1994b). In particular, the resulting reactive module does not falsify the assumptions; see also Example 4. This insight is also used by Majumdar et al. (2019) where a *non-conflicting reactive module* is directly constructed by a four-nested fixed-point algorithm over a two-player game graph which is inspired by the one in Eq. 21. Majumdar et al. (2019) further give a self-contained proof of the desired non-conflictingness result in the framework of two-player games which is slightly different from the language-based setting of Theorem 1.

Example 4 Consider the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$ discussed in Example 2, which is solved using the automaton M depicted in Fig. 1. To solve the corresponding supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ instead, an automaton $\check{M}$ conforming with Eq. 20 which corresponds to $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ is needed. 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 $\Sigma_{\text{uc}} = U$ and $\Sigma_c = Y$ in $\mathcal{A}$ and $\mathcal{G}$, these changes are only cosmetic. For convenience, we depicted $\check{M}$ corresponding to $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ in Fig. 3.

Using Eq. 21 to solve $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ over $\check{M}$ results in $\mathsf{Win}(\check{M}) = Q$ with $\mathsf{rank}(q_3) = (1, 2)$, $\mathsf{rank}(q_4) = (1, 1)$, $\mathsf{rank}(q_6) = (2, 2)$. Defining a supervisor based on this ranking and extracting a reactive module via Eqs. 26-31 results in a system strategy which always transitions from q_3 to q_4. Comparing this solution to the one obtained in Example 2, we see that the four-nested fixed point in Eq. 21 allows to distinguish between transitioning from q_3 to q_4 or to q_6 and, as a consequence, chooses the former to not falsify the assumptions. This clearly constitutes a more desirable solution to problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$.

Interestingly, $\mathcal{G}$ is relatively topologically closed w.r.t $\mathcal{A}$, and we can therefore also compute a maximally permissive supervisor (see Remark 6). This supervisor 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 $\mathcal{A}$, it will always eventually transition from q_3 to q_4, implying that the resulting closed loop fulfils the guarantee.

5.2 Supervisory control via reactive synthesis

We now consider a supervisory control problem and aim for a solution via reactive synthesis. Within this section, we discuss two different possible transformations and show that in both cases the resulting supervisor is non-conflicting if the plant behaviour $\mathcal{A}$ is topologically closed.

5.2.1 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 5.1, our approach is organised in two steps. Given a supervisory control problem, we (i) derive a corresponding reactive synthesis problem

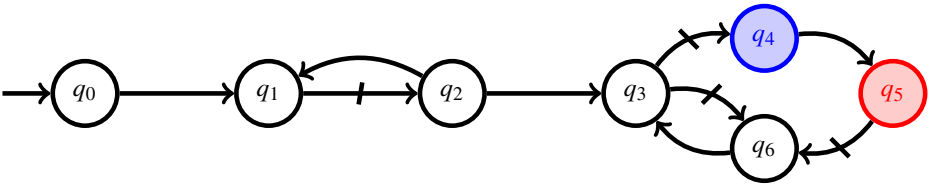

Fig. 3 Automaton $\check{M}$ representing $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ in Example 4 which corresponds to $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$ represented by M in Fig. 1. Final states are $F_{\mathcal{A}} = \{q_4\}$ (blue) and $F_{\mathcal{G}} = \{q_5\}$ (red) and transitions labelled by controllable events $\Sigma - \Sigma_{\text{uc}}$ are indicated by a tick

which we solve as outlined in Section 3.3, and (ii) convert the solution to a supervisor that solves the original supervisory control problem.

Step (i) Given a supervisory control problem $\mathsf{SCT}[\Sigma,\ \Sigma_{uc}, \mathcal{A},\ \mathcal{G}]$, we are provided an alphabet Σ, a set of uncontrollable events $\Sigma_{uc} \subseteq \Sigma$, 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 4.1 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 will pass on the first move to the system to be designed; i.e.,

$$\Sigma' := \Sigma \cup \{0\},\quad \Sigma'_{uc} := \Sigma_{uc} \cup \{0\},\qquad \Gamma' := \{\gamma \subseteq \Sigma' \mid \Sigma'_{uc} \subseteq \gamma\} \tag{32}$$

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

$$\mathcal{G}' := \{\alpha \in (\Sigma'\Gamma')^\omega \mid \mathrm{p}_{\Sigma'}\alpha \in 0\mathcal{G}\}\,. \tag{33}$$

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

$$\begin{aligned}\mathcal{A}' := \{\alpha \in (\Sigma'\Gamma')^\omega \mid \mathrm{p}_{\Sigma'}\alpha \in 0\mathcal{A} \quad &\text{and}\\ \forall\, s \in \mathrm{pfx}\alpha\,.\,\forall\,\gamma \in \Gamma'\,.\,\forall\,\sigma \in \Sigma'\,.\,s\gamma\sigma \in \mathrm{pfx}\alpha \ \to\ \sigma \in \gamma\}\,.\end{aligned} \tag{34}$$

This results in the reactive synthesis problem $\mathsf{RS}[U,\ Y, \mathcal{A}',\ \mathcal{G}']$. To this end, we assume that $\mathsf{RS}[U,\ Y, \mathcal{A}',\ \mathcal{G}']$ exhibits a solution $r : U^+ \to Y$ with associated behaviour $\mathcal{L}'$.

Step (ii) Given $\mathcal{L}'$ from Step (i), we construct

$$\begin{aligned}\mathcal{L} := \{\beta \in \Sigma^\omega \mid \exists\,\alpha \in \mathcal{L}' \quad &\text{with}\quad 0\beta = \mathrm{p}_{\Sigma'}\alpha \quad \text{and}\\ \forall\, s \in \mathrm{pfx}\alpha\,.\,\forall\,\gamma \in \Gamma'\,.\,\forall\,\sigma \in \Sigma'\,.\,s\gamma\sigma \in \mathrm{pfx}\alpha \ \to\ \sigma \in \gamma\}\,.\end{aligned} \tag{35}$$

as a candidate to solve $\mathsf{SCT}[\Sigma,\ \Sigma_{uc}, \mathcal{A},\ \mathcal{G}]$. It is shown in Appendix C, Proposition 4, that $\mathcal{L}$ indeed 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 pfx $\mathcal{L}$ and pfx $\mathcal{A}$ do not deadlock.

Referring back to Problem 2, we are left to verify 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. 16. Hence we can use topologically closedness as a sufficient condition to conclude that the supervisor with associated behaviour $\mathcal{L}$ given in Eq. 35 solves $\mathsf{SCT}[\Sigma,\ \Sigma_{uc}, \mathcal{A},\ \mathcal{G}]$.

Theorem 2 *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* $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$*. Pre-process the parameters according to* Eqs. 32-34 *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* $\mathsf{RS}[UY\mathcal{A}'\mathcal{G}']$*.*

If ***the plant*** $\mathcal{A}$ ***is topologically closed****, then* $\mathcal{L}$ *defined by* Eq. 35 *solves* $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$*. 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, this solution can be transformed into a non-blocking supervisor solving the initial control problem. Technically, the overall procedure amounts to pre-processing the problem instance by Eqs. 32-34, using the reactive synthesis procedure discussed in Section 3.3 to compute a reactive module, and post-processing the latter by Eq. 35. This is illustrated by the following example.

Example 5 Consider the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ discussed in Example 3, s.t. M depicted in Fig. 2 represents $\mathcal{G}$ and $\mathcal{A}$ with $F_{\mathcal{G}} = \{s\}$ and $F_{\mathcal{A}} = \{s, r\}$. As M cannot generate infinite strings which do not visit either r or s infinitely often, it follows that $\mathcal{A}$ is topologically closed. To obtain the corresponding reactive synthesis problem $\mathsf{RS}[UY\mathcal{A}'\mathcal{G}']$ (as defined in Eqs. 32-34), we manipulate M in Fig. 2 s.t. the resulting automaton $\check{M}$ conforms with Eq. 7 and represents $\mathcal{A}'$ and $\mathcal{G}'$.

This results in splitting every state $q \in Q$ of M into a system state q^1 and k environment states q^0_k 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^0_k is labelled with the respective control pattern $\gamma_k \subseteq \Gamma$. Outgoing transitions of q^0_k mimic outgoing transitions of q in M, i.e., a transition $(q, \sigma, q') \in \delta$ of M is copied to all $(q^0_k, \sigma, q'^1) \in \check{\delta}$ of M for which $\sigma \in \gamma_k$. Finally, we add a dummy initial state d whose outgoing transition is labelled by the dummy event 0 and which leads to the system part p^1 of M's initial state p.

The resulting automaton $\check{M}$ is depicted in Fig. 4, where all γ-labels are trimmed to the events actually available at its source state. The sets of final states are translated in the obvious way, resulting in $T^0 = \{r^0, s^0\}$ (blue) and $T^1 = \{s^1\}$ (red). Depending on the controllability status of event b we get one or two possible control patterns in state q. I.e., if $b \in \Sigma_{\text{uc}}$ we obtain the automaton without q^0_2, while $b \in \Sigma \setminus \Sigma_{\text{uc}}$ results in the full automaton containing both q^0_1 and q^0_2.

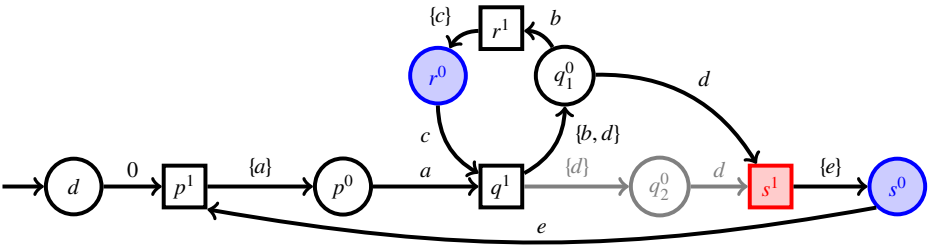

Fig. 4 Automaton $\check{M}$ representing $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$ in Example 5, which corresponds to $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ represented by M in Fig. 2 with $\Sigma_{\text{uc}} = \Sigma \setminus \{b\}$ (including q^0_2) and $\Sigma_{\text{uc}} = \Sigma$ (excluding q^0_2). The final state sets are $T^0 = \{r^0, s^0\}$ (blue) and $T^1 = \{s^1\}$ (red)

We now use $\check{M}$ as the input to the reactive synthesis algorithm in Section 3.3. If $\Sigma_{uc} = \Sigma$ 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 which coincides with the solution of case (C) in Example 3. If $\Sigma_{uc} = \Sigma \setminus \{b\}$, the system has a choice in q^1 and can apply the control pattern $\{d\}$, effectively disabling b in q of M in Fig. 2. This coincides with the solution obtained of case (B) in Example 3.

Equivalence of problem statements Given the results in Theorems 1 and 2, we have established that for topologically closed plants both synthesis problems can be solved via the respective other one. However, our construction assumes that the respective target problem exhibits a solution. Since 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 our transformation of $\mathsf{SCT}[\Sigma, \Sigma_{uc}, \mathcal{A}, \mathcal{G}]$ to $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$ retains solvability. Additionally referring to Theorem 2, this gives the following corollary.

Corollary 1 *Let Σ be a finite alphabet with the non-empty set of uncontrollable events $\Sigma_{uc} \subseteq \Sigma$, $\mathcal{A} \subseteq \Sigma^\omega$ be a plant, $\mathcal{G} \subseteq \Sigma^\omega$ be a upper bound specification and $U = \Sigma'$, $Y = \Gamma'$, $\mathcal{A}'$ and $\mathcal{G}'$ be defined by* Eqs. 32-34.

*If **the plant $\mathcal{A}$ is topologically closed**, then $\mathsf{SCT}[\Sigma, \Sigma_{uc}, \mathcal{A}, \mathcal{G}]$ has a (ω-regular) solution if and only if $\mathsf{RS}[U, Y, \mathcal{A}', \mathcal{G}']$ has a (ω-regular) solution.*

The above corollary relates to the observation in Remark 4 and Remark 8 that the algorithmic solution of both problem statements reduce to the computation of the same 2-nested fixed point if $\mathcal{A}$ is topologically closed. However, it should be noted, that the input automata to both algorithms differ (compare Figs. 2 and 4 for an example).

5.2.2 Input-output behaviours

The transformation proposed in Section 5.2.1 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 we restrict the discussion to plant behaviours in which controllable and uncontrollable events alternate. Technically, we now consider a supervisory control problem with parameters $\Sigma_{uc} \subsetneq \Sigma$ and $\mathcal{A}$, $\mathcal{G} \subseteq (\Sigma_{uc}(\Sigma - \Sigma_{uc}))^\omega$. As we will match inputs and outputs with uncontrollable and controllable events, respectively, we refer to this class of behaviours as *input-output behaviours*. We again derive a solution of the given supervisory control problem via reactive synthesis in two steps.

Step (i) Given a supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{uc}, \mathcal{A}, \mathcal{G}]$ with input-output behaviours, we can choose the correspondence $U := \Sigma_{uc}$ and $Y := \Sigma - \Sigma_{uc}$ and obtain $\mathcal{G}$, $\mathcal{A} \subseteq (UY)^\omega$; i.e., our choice constitutes qualifying parameters for the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$. Let $\mathcal{L}$ denote a solution of $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$.

Step (ii) $\mathcal{L}$ satisfies (RM1)–(RM3) and we need to derive a behaviour that satisfies (SC1) and (SC2). Topological closedness (SC1) is immediate by (RM1). Regarding (SC2), we propose the following transformation:

$$\mathcal{L}' := \mathcal{L} \cup ((\mathrm{pfx}\, \mathcal{L})(\Sigma_{uc}^\omega)) \,. \tag{36}$$

It is shown in Appendix D, Proposition 7, that the above construct $\mathcal{L}'$ indeed satisfies (SC1) and (SC2) and, moreover, retains the absence of deadlocks, i.e., pfx $\mathcal{L}'$ and 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 5.2.1, 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 3 *Given a finite alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, an **input-output plant behaviour** $\mathcal{A} \subseteq \Sigma^\omega$ and an **input-output specification behaviour** $\mathcal{G} \subseteq \Sigma^\omega$, consider the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$. 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. 36 *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 1 and Theorem 3, 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 $\mathcal{G}$ are input-output behaviours, neither of the proposed transformations affects the problem parameters $\mathcal{A}$ and $\mathcal{G}$. Hence, both problem statements are equivalent w.r.t. solvability, as summarised in the following corollary, complementing Corollary 1.

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 $\mathcal{G} \subseteq (UY)^\omega$, the supervisory control problem $\mathsf{SCT}[\Sigma, \Sigma_{\mathrm{uc}}, \mathcal{A}, \mathcal{G}]$ has a (ω-regular) solution if and only if the reactive synthesis problem $\mathsf{RS}[U, Y, \mathcal{A}, \mathcal{G}]$ has a (ω-regular) solution.*

Theorem 1 and Theorem 3 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.

In this setting, the automaton M defined for reactive synthesis and for supervisory control in Eqs. 7 and 20, respectively, coincide; compare Figs. 1 and 3 for an example. Technically, states with outgoing transitions from $U = \Sigma_{\mathrm{uc}}$ (resp. $Y = \Sigma - \Sigma_{\mathrm{uc}}$) correspond to environment states (resp. system states). Likewise, outgoing transitions from an environment state (resp. system state) are considered uncontrollable (resp. controllable). Therefore, the unconditioned version $\mathsf{Pre}(T) = \mathsf{Pre}(T, \emptyset)$ of the pre-operator defined in Eq. 22 coincides with the controllable pre-operator Pre^1 defined in Eq. 9. Following up Remarks 4 and 8 for topologically closed ω-languages $\mathcal{A}$, both synthesis algorithms compute the same 2-nested fixed-point, see Eqs. 12 and 25. Hence, for topologically closed input-output behaviours both solution techniques also coincide on the automaton level.

5.3 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 5.2 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 5.2.2.

Non-falsifiable assumptions in reactive synthesis Consider a reactive synthesis problem $\mathsf{RS}[U,\ Y,\mathcal{A},\ \mathcal{G}]$ with solution $\mathcal{L}$, which implies that $\mathcal{L}$ and $\mathcal{A}$ do not deadlock. 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 $\mathcal{L} \subseteq \mathcal{A} \rightarrow \mathcal{G}$, risk that $\alpha \notin \mathcal{G}$ (see e.g. Example 2). 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 $s \notin \mathrm{pfx}(\mathcal{A} \cap \mathcal{L})$.

This implies for all extensions $\beta \in (U \cup Y)^\omega$ with $s\beta \in \mathcal{L}$ that $s\beta \notin \mathcal{A}$. For parameters $\mathcal{G} \subseteq \mathcal{A}$ we obtain $s\beta \notin \mathcal{G}$, i.e., after the finitely many computation cycles represented by s it is known that the guarantee will not be satisfied.

This issue can be avoided if the given assumption $\mathcal{A}$ is *non-falsifiable*[11] in the following sense. Given the two player game interpretation used in the algorithmic synthesis of reactive modules (see Section 3.3), 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.

Strong non-anticipation in supervisory control A closely related issue has been identified in the context of supervisory control by 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 the following example adapted from Moor et al. (2011).

Example 6 Consider the automaton given in Figure 5 and assume that $F_{\mathcal{A}} = \{\mathsf{AA}, \mathsf{BB}\}$ and $F_{\mathcal{G}} = \{\mathsf{AA}\}$. A supervisor solving the synthesis problem may therefore at some stage disable

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

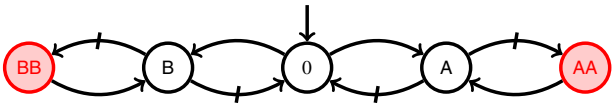

Fig. 5 Transition structure of a Büchi-automaton realising a plant behaviour that needs to anticipate future control patterns in order to satisfy its acceptance condition. Accepting states are marked in red and controllable transitions are indicated by a tick

the transition from B to BB for all future. If the plant is not aware of this restriction, it might still organise its moves in the attempt to satisfy its acceptance condition by visiting BB infinitely often, e.g., by always transitioning form 0 to B. This results in an infinite sequence which falsifies the assumptions.

To avoid the implicit need for cooperation, there is an interest in plant behaviours that can attain their acceptance conditions independently of the supervisor. A class of such plant behaviours has been characterised by Moor et al. (2011) for the special case of input-output behaviours. The reported results amount to a representation of $\mathcal{A}$ as a union of topologically closed components that each exhibit $Y = \Sigma - \Sigma_{\text{uc}}$ 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 $\text{clo}\mathcal{A}$ to equal $\text{pfx}\,\mathcal{A}$, i.e.,

$$\text{cfx}_{\text{clo}\mathcal{A},\,Y}\,\mathcal{A} = \text{pfx}\,\mathcal{A} \tag{37}$$

where $Y = \Sigma - \Sigma_{\text{uc}}$ 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 4.3, Eq. 37 requires that the *local plant* $\text{pfx}\,\mathcal{A}$ is always in a winning configuration regarding the satisfaction of its own acceptance condition, i.e., at any time the plant can decide to internally 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. 16, is implied by the absence of deadlocks.

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

$$(\text{pfx}\,\mathcal{A}) \cap (\text{pfx}\,\mathcal{L}) \subseteq \text{cfx}_{\text{clo}\mathcal{A},\,Y}\,\mathcal{A} \tag{38}$$

to conclude a non-conflicting closed loop from the absence of deadlocks; see Appendix D, Proposition 9, for a formal proof.

If we synthesise $\mathcal{L}$ by supervisory control, the closed loop will be non-conflicting by construction. To additionally ensure Eq. 38, we can pre-process the specification s.t.

$$\mathcal{G}' := \mathcal{G} \cap \{\,\alpha \in \Sigma^{\omega} \mid \text{pfx}\,\alpha \subseteq \text{cfx}_{\text{clo}\mathcal{A},\,Y}\,\mathcal{A}\,\}\,. \tag{39}$$

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 whenever an assumption is non-falsifiable we have

$\epsilon \in \text{cfx}_{\text{clo}\mathcal{A},\,Y}(\mathcal{A})$, which is indeed always the case if Eq. 37 holds; see Appendix D, Proposition 8, for a formal proof. Hence, the condition of a non-falsifiable assumption is weaker then the condition of a strongly non-anticipating plant behaviour. More precisely, we have that (i) topological closedness of $\mathcal{A}$ implies (ii) strong non-anticipation, which in turn implies (iii) (38), to finally imply (iv) $\mathcal{A}$ to be a non-falsifiable assumption. The converse implications, however, do not hold in general.

Result Referring to the simple transformation discussed in Section 5.2.2, we consider an instance of the supervisory control problem $\mathsf{SCT}[\Sigma,\ \Sigma_{\mathrm{uc}}, \mathcal{A},\ \mathcal{G}]$ where $\mathcal{A}$ and $\mathcal{G}$ are input-output behaviours, and the corresponding reactive synthesis problem $\mathsf{RS}[U,\ Y,\ \mathcal{A},\ \mathcal{G}]$, with $U = \Sigma_{\mathrm{uc}}$, $Y = \Sigma - \Sigma_{\mathrm{uc}}$. As the condition in Eq. 38 constitutes a closed-loop property, it can be verified after a solution $\mathcal{L}$ of $\mathsf{RS}[U,\ Y,\ \mathcal{A},\ \mathcal{G}]$ is computed. If this test passes, we can conclude that the absence of deadlocks, Eq. 4, implies a non-conflicting closed loop, Eq. 16. With this, we can use Eq. 38 as a sufficient condition to establish a solution of $\mathsf{SCT}[\Sigma,\ \Sigma_{\mathrm{uc}}, \mathcal{A},\ \mathcal{G}]$ via $\mathsf{RS}[U,\ Y,\ \mathcal{A},\ \mathcal{G}]$ analogously to Theorem 3. Nevertheless, as Eq. 38 is weaker then topological closeness of $\mathcal{A}$, we obtain a generalisation of Theorem 3.

Theorem 4 *Given the premises of Theorem 3,*
$\mathcal{L}'$ *defined by* Eq. 36 *solves* $\mathsf{SCT}[\Sigma,\ \Sigma_{\mathrm{uc}},\ \mathcal{A},\ \mathcal{G}]$ *if* Eq. 38 *holds. If* $\mathcal{L}'$ *is* ω*-regular, then so is* $\mathcal{L}$*.*

Again referring to Theorem 1 from Section 5.1, but now using Theorem 4 rather than Theorem 3, we obtain the following generalisation of Corollary 2.

Corollary 3 *Given an alphabet* Σ *with* $\emptyset \neq \Sigma_{\mathrm{uc}} \subsetneq \Sigma$*, let* $U = \Sigma_{\mathrm{uc}}$ *and* $Y = \Sigma - \Sigma_{\mathrm{uc}}$*. 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* (ω*-regular) solution if and only if the reactive synthesis problem* $\mathsf{RS}[U,\ Y, \mathcal{A},\ \mathcal{G}]$ *has a* (ω*-regular) solution.*

Referring back to the discussion below Corollary 2, we still have that the automata realisations of the two corresponding problem instances are effectively identical. As $\mathcal{A}$ is not assumed to be topologically closed, we now have to use the three-nested fixed-point algorithm in Eq. 10 to solve the reactive synthesis problem and, in turn, to obtain a solution of supervisory control problem. Note that it is not evident how the four-nested fixed-point for supervisory controller synthesis (21) can be directly converted to a three-nested fixed-point to specifically address strongly non-anticipating plants.

Example 7 Consider the instance of Problem 2 represented by $\breve{M}$ depicted in Fig. 3 and the automaton M depicted in Fig. 1 representing the corresponding instance of Problem 1. Recall from Example 2 that a solution of the latter is given by a reactive module which always transitions from from q_3 to q_6. The only possible closed-loop run on M is hence given by the sequence $q_0q_1q_2(q_3q_6)^{\omega}$, and this run generates an ω-word $\alpha \in (\mathrm{clo}\mathcal{A}) \cap \mathcal{L}$ but $\alpha \notin \mathcal{A}$. In particular, we have $\mathrm{pfx}\,\alpha \subseteq (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L})$ and, if our condition in Eq. 38 was satisfied, we had that $\mathrm{pfx}\,\alpha \subseteq \mathrm{cfx}_{\mathrm{clo}\mathcal{A},\,Y}\mathcal{A}$. 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, Eq. 38 is not satisfied for this example. This is as expected, since the given solution to Problem 1 falsifies the assumptions.

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, matches 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 5.1 (Theorem 1). 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 2, topologically closed plants with alternating inputs and outputs, Theorem 3, and strongly non-anticipating plants with alternating inputs and outputs, Theorem 4. 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 synthesise 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 1 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 3 and 4. 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 the anonymous reviewers for their valuable comments and suggestions that helped us to improve our paper.

Funding Information Open access funding provided by Max Planck Society.

Appendix

A behavioural characterisations of the respective problem statements

We provide proofs for the behavioural characterization of Problems 1 and 2 in Lemma 1 and 2, respectively.

Proof of Lemma 1 We prove both implications separately. First, let $r : U^+ \to Y$ be a reactive module with associated behaviour $\mathcal{L} \subseteq (UY)^\omega$ as in Eq. 2. For our argument, we consider the $*$-language

$$L := \{\, t \in (UY)^* \mid \forall\, s \in (U \cup Y)^* \,.\, \forall\, y \in Y \,.\, sy \leq t \;\to\; y = r(\mathrm{p}_U s) \,\}\,, \tag{40}$$

to observe $(\mathrm{pfx}\,\mathcal{L}) \cap (UY)^* \subseteq L$. By $\mathcal{L} \subseteq (UY)^\omega$, this implies $\mathcal{L} \subseteq \mathrm{limpfx}\,\mathcal{L} \subseteq \mathrm{lim}L$. Moreover, $t \in L$ implies $(\mathrm{pfx}t) \cap (UY)^* \subseteq L$ and we obtain $\mathrm{lim}L \subseteq \mathcal{L}$. We conclude $\mathrm{lim}L = \mathcal{L}$, and we will use this preliminary observation to establish (RM1)–(RM3).

Ad (RM2)

Ad (RM1) We have that $\mathrm{clo}\mathcal{L} = \mathrm{limpfx}\,\mathcal{L} \subseteq \mathrm{lim}L = \mathcal{L}$, and, hence, $\mathcal{L}$ is topologically closed.

Ad (RM2) We pick arbitrary $s \in (\mathrm{pfx}\,\mathcal{L}) \cap (UY)^*$, $u \in U$, and show that $su \in \mathrm{pfx}\,\mathcal{L}$. Let $t_1 := s$ and define the sequence $(t_i)_{i\in\mathbb{N}}$ by $t_{i+1} := t_i u r(\mathrm{p}_U(t_i u))$ to observe $t_i \in L$ for all $i \in \mathbb{N}$. By construction, $(t_i)_{\mathbb{N}}$ is strictly monotone and, hence the limit amounts to a singleton $\alpha \in (UY)^\omega$, $\{\,\alpha\,\} = \mathrm{lim}\{\, t_i \mid i \in \mathbb{N}\,\} \subseteq \mathrm{lim}\, L = \mathcal{L}$. Observe that $su < t_2 < \alpha$, to conclude $su \in \mathrm{pfx}\,\mathcal{L}$. As u was chosen arbitrarily, and since $su' \in \mathrm{pfx}\,\mathcal{L}$ with $u' \in U$ implies $s \in (UY)^*$, this constitutes (RM2).

Ad (RM3) Pick any $s \in \mathrm{pfx}\,\mathcal{L}$ and any $y', y'' \in Y$ such that $sy', sy'' \in \mathrm{pfx}\,\mathcal{L}$. We need to show that $y' = y''$. As $sy', sy' \in \mathrm{pfx}\,\mathcal{L}$, we have $sy', sy' \in L$ and, hence, $y' = r(\mathrm{p}_U s) = y''$.

Ad $\mathcal{L} \neq \emptyset$ Observe that $\epsilon \in L$. Hence, we can pick an arbitrary $u \in U$ and do the same construction as in (RM2), now staring with $t_1 = \epsilon$, to obtain $\alpha \in \mathcal{L}$. □

Thus, we have established (RM1)–(RM3) and non-emptiness for $\mathcal{L}$ defined by Eq. 2.

For the converse implication of the lemma, consider any non-empty $\mathcal{L} \subseteq (UY)^\omega$ that complies with (RM1)–(RM3). For a proof, we need to construct a reactive module such that the associated behaviour matches $\mathcal{L}$. This is done in three steps (A)–(C).

Step (A) To observe that for any $v \in U^+$ there uniquely exist $s \in (UY)^*U$ and $y \in Y$ such that

$$\mathrm{p}_U s = v \quad \text{and} \quad sy \in \mathrm{pfx}\,\mathcal{L}\,, \tag{41}$$

we argue by induction over the length of v. For $v \in U$, i.e., length 1, non-emptiness of $\mathcal{L}$ and (RM2) implies $v \in \mathrm{pfx}\,\mathcal{L}$. In particular, we can choose $y \in Y$ such that $vy \in \mathrm{pfx}\,\mathcal{L}$. By (RM3), the latter choice of y is unique. 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 (41) for v. By (RM2), we obtain $syu \in \text{pfx}\,\mathcal{L}$, and, hence, we can choose $y' \in Y$ such that $syuy' \in \text{pfx}\,\mathcal{L}$.

With $s' = syu$, we observe $\text{p}_U s' = vu$ and $s'y' \in \text{pfx}\,\mathcal{L}$, i.e., s' and y' satisfy Eq. 41 for vu. Now consider any $s'' \in (U \cup Y)^*$ and $y'' \in Y'$ with $\text{p}_U s'' = vu$ and $s''y'' \in \text{pfx}\,\mathcal{L}$. Since the length of vu is at least 2, we can decompose by $s'' = t\hat{u}\hat{y}u$ with $\hat{u} \in U$ and $\hat{y} \in Y$. By the induction hypothesis, we obtain $t\hat{u} = s$ and $\hat{y} = y$. Hence, $s'' = syu = s'$. Together with (RM3), this also implies $y'' = y'$.

Step (B) Referring to Step (A), we can define functions $f' : U^+ \to (U \cup Y)^*$ and $r' : U^+ \to Y$ such that Eq. 41 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.

Step (C) We now show that the behaviour $\mathcal{L}'$ associated with 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 $v = \text{p}_U s$. By $sy \in \text{pfx}\,\mathcal{L}$ and the unique choice in (A) this implies $s = f'(v)$ and $y = r'(v)$. Since this holds true for any prefix, we have $\alpha \in \mathcal{L}'$. For the converse inclusion, pick any $\alpha \in \mathcal{L}'$. 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 $s_1 \in \text{pfx}\,\mathcal{L}$. Now assume $s_i \in \text{pfx}\,\mathcal{L}$ for some $i \in \mathbb{N}$ and consider $v := \text{p}_U s_i$. By the claim in (A), there uniquely exists $s \in (UY)^*U$ and $y \in Y$ such that Eq. 41 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 behaviour, we also have $y_i = r'(v)$. We conclude $s_i y_i \in \text{pfx}\,\mathcal{L}$, and hence, by (RM2), $s_{i+1} \in \text{pfx}\,\mathcal{L}$. Thus, infinitely many prefixes $(s_i)_{i\in\mathbb{N}}$ of α are within $\text{pfx}\,\mathcal{L}$. Topological closedness (RM1) then implies $\alpha \in \mathcal{L}$. □

Proof of Lemma 2 Technically, our lemma can be interpreted as a special case of Proposition 3.1 in Ramadge (1989a) when applied to the formal plant behaviour $\mathcal{A} = \Sigma^\omega$. In the interest of a self-contained exposition, we provide a direct proof.

First, let $f : \Sigma^* \to \Gamma$ denote a supervisor with associated behaviour $\mathcal{L}$ as in Eq. 14. For our argument, we consider the $*$-language

$$L := \{\, t \in \Sigma^* \mid \forall\, s \in \Sigma^* \,.\, \forall\, \sigma \in \Sigma \,.\, s\sigma \leq t \;\to\; \sigma \in f(s) \,\}\,. \tag{42}$$

We observe that $\text{pfx}\,\mathcal{L} \subseteq L$ and, hence, $\mathcal{L} \subseteq \text{limpfx}\,\mathcal{L} \subseteq \text{lim}L$. Moreover, $t \in L$ implies $\text{pfx}t \subseteq L$, i.e., L is prefix-closed, and $\text{lim}L \subseteq \mathcal{L}$. Hence, $\text{lim}L = \mathcal{L}$. We now establish (SC1), (SC2) and non-emptiness of $\mathcal{L}$.

Ad (SC1) As the limit of a prefix-closed $*$-language, $\mathcal{L}$ is topologically closed.

Ad (SC2) We pick arbitrary $s \in \text{pfx}\,\mathcal{L}$ and $\sigma \in \Sigma_{\text{uc}}$ to show that $s\sigma \in \text{pfx}\,\mathcal{L}$. We begin with $t_1 := s \in L$ to construct a sequence $(t_i)_{i\in\mathbb{N}}$ by $t_{i+1} := t_i\sigma$. From $\sigma \in \Sigma_{\text{uc}} \subseteq f(t_i)$, we obtain $t_i \in L$ for all $i \in \mathbb{N}$. By construction, $(t_i)_{i\in\mathbb{N}}$ is strictly monotone and, hence, the limit amounts to a singleton $\alpha \in \Sigma^\omega$ with $\{\alpha\} = \text{lim}\{\, t_i \mid i \in \mathbb{N} \,\} \subseteq \text{lim}L = \mathcal{L}$. Observe that $s\sigma = t_2 < \alpha$, to conclude $s\sigma \in \text{pfx}\,\mathcal{L}$.

Ad $\mathcal{L} \neq \emptyset$ Observe that $\epsilon \in L$. Since $\Sigma_{\text{uc}} \neq \emptyset$ we can pick an arbitrary $\sigma \in \Sigma_{\text{uc}}$ and conduct the same construction as in (SC2), now starting with $t_1 = \epsilon$, to obtain $\alpha \in \mathcal{L}$.

Thus, we have established (SC1), (SC2) and non-emptiness for $\mathcal{L}$ defined by Eq. 14.

For the converse implication, we now consider any non-empty $\mathcal{L} \subseteq (UY)^\omega$ that complies with (SC1) and (SC2) to define the supervisor $f' : \Sigma^* \to \Gamma$,

$$f'(s) := \{\, \sigma \in \Sigma \mid s\sigma \in \text{pfx}\,\mathcal{L} \,\} \cup \Sigma_{\text{uc}}\,, \tag{43}$$

with associated behaviour $\mathcal{L}'$, and we show that $\mathcal{L} = \mathcal{L}'$.

Ad $\mathcal{L} \subseteq \mathcal{L}'$ 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 $s\sigma \in \text{pfx}\,\mathcal{L}$ and, hence, $\sigma \in f'(s)$. By the arbitrary choice of the prefix, we conclude that $\alpha \in \mathcal{L}'$ from Eq. 14.

Ad $\mathcal{L}' \subseteq \mathcal{L}$ Pick any $\alpha \in \mathcal{L}'$. We fist show that $\text{pfx}\,\alpha \subseteq \text{pfx}\,\mathcal{L}$ by induction over the length of the respective prefix. Clearly, the claim is true for the prefix of length 0, i.e., we have $\epsilon \in \text{pfx}\,\mathcal{L}$ by non-emptiness of $\mathcal{L}$. For the induction step, consider $s \in \text{pfx}\,\mathcal{L}$ and $\sigma \in \Sigma$ with $s\sigma < \alpha$. Since $\alpha \in \mathcal{L}'$ we obtain $\sigma \in f'(s)$ from Eq. 14. If σ is in the left union component of Eq. 43, we directly obtain $s\sigma \in \text{pfx}\,\mathcal{L}$. If σ is in the right union component Σ_{uc}, we also obtain $s\sigma \in \text{pfx}\,\mathcal{L}$ from the proof of (SC2). Hence, $\text{pfx}\,\alpha \subseteq \text{pfx}\,\mathcal{L}$ and we finally obtain $\alpha \in \mathcal{L}$ by topological closeness (SC1). □

B Reactive synthesis via supervisory control

The following technical proposition summarises relevant properties obtained by our construction of $\mathcal{L}''$ in Eqs. 26, 27 and 31.

Proposition 2 *Given non-empty alphabets U, Y, Σ and Σ_{uc}, where $\Sigma = U \dot\cup Y$ and $\Sigma_{\text{uc}} = U$, consider any supervisor $f : \Sigma^* \to \Gamma$ with associated behaviour $\mathcal{L}$ that solves $\mathsf{SCT}[\Sigma, \Sigma_{\text{uc}}, \mathcal{A}, \mathcal{G}]$ for parameters $\mathcal{A}, \mathcal{G} \subseteq (UY)^\omega$.*

Then the behaviour $\mathcal{L}''$ defined by Eqs. 26, 27 and 31*, exhibits properties (RM1)–(RM3). 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 Recall that $\mathcal{L}'$ is the behaviour associated with the supervisor f' and, hence, exhibits the properties (SC1) and (SC2). As a preliminary observation, we show that $\mathcal{L}'$ and $(UY)^\omega$ are non-conflicting. Pick any $s \in (\text{pfx}\,\mathcal{L}') \cap (\text{pfx}(UY)^\omega)$. If $s \in (UY)^*$, we refer to universal controllability (SC2) to obtain $su \in \text{pfx}\,\mathcal{L}'$ for any $u \in U$. If $s \notin (UY)^*$, we must have $s \in (UY)^*U$ and we refer to Eq. 29 to obtain that $sy \in \text{pfx}\,\mathcal{L}'$ for some $y \in Y$. In both cases, we have established the existence of some $\sigma \in \Sigma$ such that $s\sigma \in (\text{pfx}\,\mathcal{L}') \cap (\text{pfx}(UY)^\omega)$, i.e., the two languages do not deadlock. Note that $\mathcal{L}'$ is topologically closed (SC1) and that $(UY)^\omega$ is topologically closed, too. Thus, not to deadlock implies non-conflictingness.

We now turn to the individual claims suggested by the proposition.

Ad (RM1) The intersection $\mathcal{L}''$ of two topologically closed languages is itself topologically closed.

Ad (RM2) Pick any $s \in \Sigma^*, u', u'' \in U$ and assume that $su' \in \text{pfx}\,\mathcal{L}''$. In particular, $su' \in \text{pfx}\,\mathcal{L}'$ and, by (SC2), $su'' \in \text{pfx}\,\mathcal{L}'$. By $su' \in \text{pfx}\,\mathcal{L}'' \subseteq \text{pfx}((UY)^\omega)$, we obtain $su'' \in \text{pfx}((UY)^\omega)$. Since $\mathcal{L}'$ and $(UY)^\omega$ are non-conflicting, this implies $su'' \in \text{pfx}(\mathcal{L}'' \cap (UY)^\omega) = \text{pfx}\,\mathcal{L}''$.

Ad (RM3) Pick any $s \in \Sigma^*, y', y'' \in Y$ and assume that $sy' \in \text{pfx}\,\mathcal{L}''$. $sy'' \in \text{pfx}\,\mathcal{L}''$. This implies $sy' \in \text{pfx}\,\mathcal{L}'$ and $sy'' \in \text{pfx}\,\mathcal{L}'$, and, by Eq. 30, $y' = y''$.

Ad $\mathcal{A} \cap \mathcal{L}'' \subseteq \mathcal{A} \cap \mathcal{L}$ We pick any $\alpha \in \mathcal{A} \cap \mathcal{L}''$ and establish $\alpha \in \mathcal{L}$. Since $\mathcal{L}$ is topologically closed, it is sufficient to show that $\text{pfx}\alpha \subseteq \text{pfx}\,\mathcal{L}$ and we do so by induction. By $\mathcal{L} \neq \emptyset$, we have $\epsilon \in \text{pfx}\,\mathcal{L}$, i.e., the claim is true for the prefix $\epsilon < \alpha$ of length 0. Now consider $s \in \Sigma^*, \sigma \in \Sigma$ with $s\sigma < \alpha$ and assume that $s \in \text{pfx}\,\mathcal{L}$. If $s \in (UY)^*$, we have that $\sigma \in U$ and refer to (RM2) to obtain $s\sigma \subseteq \text{pfx}\,\mathcal{L}$. If $s \notin (UY)^*$, we must have $s \in (UY)^*U$, and, hence, $\sigma \in f'(s) \cap Y$. Denote $Y_{\text{f}}(s)$ the outputs that comply with the original supervisor f given s, i.e., $Y_{\text{f}}(s) := f(s) \cap Y = \{\, y \in Y \mid sy \in \text{pfx}\,\mathcal{L} \,\}$. Likewise, denote Y_{a} the outputs that comply with the plant given s, i.e., $Y_{\text{a}}(s) := \{\, y \in Y \mid sy \in \text{pfx}\,\mathcal{A} \,\}$. Then non-conflictingness of $\mathcal{A}$ and $\mathcal{L}$ implies $Y_{\text{a}}(s) \cap Y_{\text{f}}(s) \neq \emptyset$. By

the definition of the post-processed supervisor h, we have $h(s) \cap Y = (Y_a(s) \cap Y_f(s))$. By the definition of f', we have that $f'(s) \cap Y$ is a singleton, i.e., we obtain that $\{\sigma\} = f'(s) \cap Y \subseteq h(s) \cap Y = Y_a(s) \cap Y_f(s)$. In particular, this implies $\sigma \in f(s)$. We extend $s\sigma$ by an arbitrary $\beta \in \Sigma_{uc}^{\omega}$, and refer to the induction hypothesis $s \in \text{pfx}\,\mathcal{L}$ to obtain $s\sigma\beta \in \mathcal{L}$. This implies $s\sigma \in \text{pfx}\,\mathcal{L}$ and thereby completes the induction step. This concludes the proof of $\alpha \in \mathcal{L}$ with $\mathcal{A} \cap \mathcal{L}'' \subseteq \mathcal{A} \cap \mathcal{L}$ as an immediate consequence.

Ad $\mathcal{A}$ and $\mathcal{L}''$ not to deadlock Pick any $s \in (\text{pfx}\,\mathcal{A}) \cap (\text{pfx}\,\mathcal{L}'')$. If $s \in (UY)^*$, we refer to (RM2) to obtain $sU \subseteq \text{pfx}\,\mathcal{L}''$. Since $\mathcal{A} \subseteq (UY)^{\omega}$, there must exist some $u \in U$ such that $su \in \text{pfx}\,\mathcal{A}$. Thus, we have $su \in (\text{pfx}\,\mathcal{A}) \cap (\text{pfx}\,\mathcal{L}'')$. If $s \notin (UY)^*$, we must have $s \in (UY)^*U$. Consider the same sets of enabled output events as above, i.e., $Y_f := f(s) \cap Y = \{\, y \in Y \mid sy \in \text{pfx}\,\mathcal{L} \,\}$ and $Y_a := \{\, y \in Y \mid sy \in \text{pfx}\,\mathcal{A} \,\}$, and recall that $Y_a \cap Y_f \neq \emptyset$. By the definition of the post-processed supervisor h, we have $h(s) = \Sigma_{uc} \cup (Y_a \cap Y_f)$. Thus, $y \in f'(s)$ for some $y \in Y_a \cap Y_f$. As above, we extend sy by an arbitrary $\beta \in \Sigma_{uc}^{\omega}$ to obtain $sy\beta \in \mathcal{L}'$, and, hence $sy \in \text{pfx}\,\mathcal{L}'$. Since $\mathcal{L}'$ and $(UY)^{\omega}$ are non-conflicting, we finally obtain that $sy \in (\text{pfx}\,\mathcal{L}') \cap (\text{pfx}((UY)^{\omega})) = \text{pfx}\,\mathcal{L}''$. □

Proof of Theorem 1 We refer to the above proposition and obtain that $\mathcal{L}''$ satisfies (RM1)–(RM3) 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, $\mathcal{L}'' \subseteq \mathcal{G} \cup (\Sigma^{\omega} - \mathcal{A}) = \mathcal{A} \rightarrow \mathcal{G}$. 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 h is the intersection of clo$A$ and $\mathcal{L}$ and amounts to a product composition. The static filter h' can be implemented on a per-state basis. In particular, the language $\mathcal{L}'$ associated with $f' := h' \circ h$ is ω-regular. Hence, $\mathcal{L}'' = \mathcal{L}' \cap (UY)^{\omega}$ is ω-regular, too. □

C Supervisory control via reactive synthesis – control-patterns as outputs

The following three technical propositions summarise relevant properties obtained by our construction of $\mathcal{L}$ from $\mathcal{L}'$ in Eq. 35 w.r.t. the plant and specification as transformed by Eqs. 33 and 34.

Proposition 3 *Consider a finite alphabet* Σ, *uncontrollable events* Σ_{uc}, $\emptyset \neq \Sigma_{uc} \subseteq \Sigma$, *control patterns* $\Gamma := \{\, \gamma \subseteq \Sigma \mid \Sigma_{uc} \subseteq \gamma \,\}$ *and extended variants thereof as defined in* Eq. 32, *and, for an arbitrary language* $\mathcal{L}' \subseteq (\Sigma'\Gamma')^{\omega}$, *the variant* $\mathcal{L} \subseteq \Sigma^{\omega}$ *defined in* Eq. 35. *If* $\mathcal{L}'$ *is topologically closed, then so is* $\mathcal{L}$.

Proof In general, topological closedness is not retained under projection. Thus, in order to establish the claim, we need to explicitly refer to the special situation of alternating visible symbols and invisible symbols, as well as to the fact that the alphabets under consideration are finite. Pick an arbitrary $\beta \in \text{clo}\mathcal{L}$ and denote $(s_i)_{i\in\mathbb{N}}$ a sequence $s_i < s_{i+1} \in \text{pfx}\beta \subseteq \text{pfx}\,\mathcal{L}$ 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

$$R_i := \{\, r \in (\Sigma'\Gamma')^* \mid \exists\, \alpha \in (\Sigma'\Gamma')^{\omega} \text{ with } r\alpha \in \mathcal{L}' \text{ such that } \mathrm{p}_{\Sigma'} r = 0 s_i \text{ and } \forall\, \gamma \in \Gamma' \,.\, \sigma \in \Sigma' \,.\, s\gamma\sigma \in \text{pfx}r \rightarrow \sigma \in \gamma \,\} \subseteq \text{pfx}\,\mathcal{L}' \,. \tag{44}$$

Note that $s_i \in \mathrm{pfx}\,\mathcal{L}$ 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:

$$\forall\, i,\ j \in \mathbb{N}\,.\,\forall\, t \in R_j\,.\,\exists\, r \in R_i\,.\, i \leq j \ \rightarrow\ r \leq t\,; \tag{45}$$

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

$$R_{i,j} := \{\, r \in R_i \mid \exists\, t \in R_j\,.\, r \leq t\,\}\,, \tag{46}$$

where $i,\ j \in \mathbb{N},\ i \leq j$. By Eq. 45, $R_{i,j}$ 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

$$N_i := \bigcap_{j \geq i} R_{i,j} \ \subseteq\ R_i \tag{47}$$

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

$$\forall\, i \in \mathbb{N}\,.\,\forall\, r \in N_i\,.\,\exists\, t \in N_{i+1}\,.\, r \leq t \tag{48}$$

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. 48. By the latter property, we begin with an arbitrary $r_1 \in N_1$ and successively obtain a sequence $(r_i)_{i \in \mathbb{N}}$, $r_i \in N_i \subseteq \mathrm{pfx}\,\mathcal{L}'$, $r_i < r_{i+1}$ 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 of $\mathcal{L}'$, also $\alpha \in \mathcal{L}'$. Referring to the second condition in Eq. 44, we also obtain that $\sigma \in \gamma$ for all $\gamma \in \Gamma'$, $\sigma \in \Sigma'$ and s with $s\gamma\sigma \in \mathrm{pfx}\alpha$, and, hence $\beta \in \mathcal{L}$. Thus, $\mathcal{L}$ is topologically closed. □

Proposition 4 *Consider a finite alphabet* Σ, *uncontrollable events* Σ_{uc}, $\emptyset \neq \Sigma_{\mathrm{uc}} \subseteq \Sigma$, *control patterns* $\Gamma := \{\, \gamma \subseteq \Sigma \mid \Sigma_{\mathrm{uc}} \subseteq \gamma\,\}$ *and extended variants thereof as defined in* Eq. 32, *and, for an arbitrary language* $\mathcal{L}' \subseteq (\Sigma'\Gamma')^\omega$, *the variant* $\mathcal{L} \subseteq \Sigma^\omega$ *defined in* Eq. 35.

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. 33 and 34. *If* $\mathcal{L}' \subseteq \mathcal{A}' \rightarrow \mathcal{G}'$ *then* $\mathcal{L} \subseteq \mathcal{A} \rightarrow \mathcal{G}$.

Proof Note that topological closedness (SC1) is provided by Proposition 3 and we are left to prove the remaining claims suggested by the proposition.

Ad (SC2) In order to establish controllability, pick any $s \in \mathrm{pfx}\,\mathcal{L}$ and $\sigma \in \Sigma_{\mathrm{uc}}$ and choose $\beta \in \Sigma^\omega$ such that $s\beta \in \mathcal{L}$. By the definition Eq Eq. 35 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 (RM2), we extend $r_1 \in \mathrm{pfx}\,\mathcal{L}'$ by σ to obtain $r_1\sigma \in \mathrm{pfx}\,\mathcal{L}'$. 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 $r_2 := r_1\sigma\gamma \in \mathrm{pfx}\,\mathcal{L}'$ 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, pfx $\alpha' \subseteq$ 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, $0s\sigma = \mathrm{p}_{\Sigma'}r_2 \in 0\mathrm{pfx}\beta'$ and, hence, $s\sigma \in$ pfx $\mathcal{L}$.

Ad $\mathcal{L} \neq \emptyset$ We refer to $\mathcal{L}' \neq \emptyset$ and (RM2) to observe that $0 \in$ pfx $\mathcal{L}'$. Thus, we can pick $r_1 \in 0\Gamma'$ such that $r_1 \in$ pfx $\mathcal{L}'$. Now assume that $r_i \in$ 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 $s\gamma\sigma \in$ pfx $\mathcal{L}'$ by (RM2). Hence, there exists $r_{i+1} \in r_i\sigma\Gamma'$ such that $r_{i+1} \in$ pfx $\mathcal{L}' \cap (0\Gamma')(\Sigma\Gamma')^*$. This establishes a monotone sequence $(r_i)_{i\in\mathbb{N}}$, $r_i < r_{i+1} \in$ 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, $\beta \in \mathcal{L}$.,

Ad $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$ Pick any $\beta \in \mathcal{L}$ and choose $\alpha \in \mathcal{L}'$ according to Eq. 35, 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. 34. Since $\mathcal{L}'$ solves $\mathsf{RS}[\Sigma', \Gamma', \mathcal{A}', \mathcal{G}']$ this implies $\alpha \in \mathcal{G}'$, and, by Eq. 33, $0\beta = \mathrm{p}_{\Sigma'}\alpha \in 0\mathcal{G}$. Thus, we observe that $\beta \in \mathcal{G}$. □

Proposition 5 *Consider a finite alphabet* Σ*, uncontrollable events* Σ_{uc}*,* $\emptyset \neq \Sigma_{\mathrm{uc}} \subseteq \Sigma$*, control patterns* $\Gamma := \{\gamma \subseteq \Sigma \mid \Sigma_{\mathrm{uc}} \subseteq \gamma\}$ *and extended variants* Σ' *and* Γ' *thereof as defined in* Eq. 32*. Given a plant* $\mathcal{A} \subseteq \Sigma^\omega$*, construct the assumption* $\mathcal{A}' \subseteq (\Sigma'\Gamma')^\omega$ *by* Eq. 34*. For a reactive module* $\mathcal{L}' \subseteq (UY)^\omega$ *with input range* $U := \Sigma'$*, output range* $Y := \Gamma'$ *and satisfying (RM1) and (RM2), consider the supervisor candidate* $\mathcal{L} \subseteq \Sigma^\omega$ *defined in* Eq. 35*. If* pfx $\mathcal{A}'$ *and* pfx $\mathcal{L}'$ *do not deadlock, then neither do* pfx $\mathcal{A}$ *and* pfx $\mathcal{L}$*.*

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

We first refer to $s \in$ 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. 35, 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 pfx $\alpha \subseteq L$, where

$$L := \mathrm{pfx}\{t \in (\Sigma'\Gamma')^*) \mid \forall\, \sigma \in \Sigma\,.\,\forall\, \gamma \in \Gamma\,.\,\forall\, r \in \mathrm{pfx}t\,.\, r\gamma\sigma \leq t \;\rightarrow\; \sigma \in \gamma\}\,. \tag{49}$$

In particular, we can rewrite

$$\alpha = 0\gamma_1\sigma_1\gamma_2\sigma_2 \cdots \gamma_n\sigma_n\gamma_{n+1}\sigma_{n+1}\cdots\,, \tag{50}$$

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 $s \in$ 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

$$\alpha' := 0\gamma_1\sigma_1\gamma_2\sigma_2 \cdots \gamma_n\sigma_n\gamma'_{n+1}\sigma'_{n+1}\cdots\,. \tag{51}$$

Again, we have pfx $\alpha' \subseteq L$ 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 $0t \in$ (pfx $\mathcal{A}'$) $\cap$ (pfx $\mathcal{L}'$). Since $\mathcal{A}'$ and $\mathcal{L}'$ do not deadlock, we can extend $0t$ by $\sigma'' \in \gamma'' \subseteq \Gamma$ to obtain $0t\gamma''\sigma'' \in$ (pfx $\mathcal{A}'$) $\cap$ (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, $s\sigma'' \in$ pfx $\mathcal{A}$.

We can also extend $0t\gamma''\sigma''$ to an ω-word in $\mathcal{L}'$, however, in order to address (35) we need to take care that all prefixes comply with past control patterns. For an inductive argument, pick any $r\sigma \in \text{pfx}\,\mathcal{L}'$. By the alternating structure of $\mathcal{L}'$, we find $\gamma \in \Gamma$ such that $r\sigma\gamma \in \text{pfx}\,\mathcal{L}'$ and, by the free input (RM2), we can choose any $\varrho \in \gamma$ to obtain $r\sigma\gamma\varrho \in \text{pfx}\,\mathcal{L}'$. This construction maintains compliance with control patterns, i.e., $r\sigma \in L$ implies $r\sigma\gamma\varrho \in L$. Applying this construction to the initial string $0t\gamma''\sigma'' \in \text{pfx}\,\mathcal{L}'$, and referring to topological closedness (RM1), we obtain an ω-word $0t\gamma''\sigma''\alpha''' \in \mathcal{L}'$ such that $\text{pfx}(0t\gamma''\sigma''\alpha''') \subseteq L$. For the projection, we have $\text{p}_{\Sigma'}(0t\gamma''\sigma''\alpha''') = 0s\sigma''\text{p}_{\Sigma'}\alpha$, and, referring to the definition of $\mathcal{L}$ in Eq. 35, we obtain $s\sigma'' \in \text{pfx}\,\mathcal{L}$. This concludes the proof. □

Proof of Theorem 2 Regarding $\mathcal{L}$, (SC1), (SC2) and non-emptyness are established by Propositions 3 and 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. Since the intersection of non-empty and non-conflicting languages is non-empty[12], we obtain for the closed-loop behaviour $\emptyset \neq \mathcal{K} := \mathcal{A} \cap \mathcal{L}$. 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 $A' = (Q,\ \Sigma' \cup \Gamma',\ \delta,\ Q_0)$. 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 finite automaton realisation A of $\mathcal{L}$ as defined by Eq. 35 can be constructed in three steps.

Step (i) We can test on a per state basis whether or not enabled transitions are labeled in compliance with the recent control pattern. Here, we may need to split states in order for the most recent control pattern to be unique for each state. We then remove all transitions which are meant to be disabled by the most recent control pattern an denote the resulting automaton A''. This automaton realises the language

$$\mathcal{L}'' := \{\, \alpha \in \mathcal{L}' \mid \forall\, s \in \text{pfx}\,\alpha\ .\ \forall\, \gamma \in \Gamma'\ .\ \forall\, \sigma \in \Sigma'\ .\ s\gamma\sigma \in \text{pfx}\alpha\ \rightarrow\ \sigma \in \gamma\,\}\,. \tag{52}$$

Step (ii) We refer to well known algorithms that implement the projection to obtain A''' to realise $\mathcal{L}''' := \text{p}_{\Sigma'}\mathcal{L}''$ and to observe that

$$\mathcal{L}''' := \{\, \beta \in \Sigma^\omega \mid \exists\, \alpha \in \mathcal{L}' \text{ with } \beta = \text{p}_{\Sigma'}\alpha \text{ and } \forall\, s \in \text{pfx}\,\alpha\ .\ \forall\, \gamma \in \Gamma'.\forall\, \sigma \in \Sigma'\ .\ s\gamma\sigma \in \text{pfx}\alpha \rightarrow\ \sigma \in \gamma\,\}\,. \tag{53}$$

Although in general projection does not retain realisability by deterministic Büchi automata, in the absence of an acceptance condition we may determinise the result by the common subset-construction.

Step (iii) 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. We denote the resulting automaton A and observe that A indeed realises $\mathcal{L}$. □

The following proposition is used to support the proof of the equivalence Corollary 1.

[12] If $\mathcal{M}, \mathcal{N} \subseteq \Sigma^\omega$ are both non-empty and non-conflicting, we have that $\epsilon \in (\text{pfx}\,\mathcal{M}) \cap (\text{pfx}\,\mathcal{N}) = \text{pfx}(\mathcal{M} \cap \mathcal{N})$ and, hence, $\mathcal{M} \cap \mathcal{N} \neq \emptyset$.

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 a 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*

$$\mathcal{L}' := \{\, \alpha \in (\Sigma'\Gamma')^\omega \mid \forall\, \gamma \in \Gamma' \,.\, \forall s \in \mathrm{pfx}\,\alpha \,.\, s\gamma < \alpha \rightarrow \gamma = f(\mathrm{p}_\Sigma s) \cup \{0\} \,\}\,, \tag{54}$$

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* (32)-(34). *If $\mathcal{L}$ is ω-regular, then so is $\mathcal{L}'$.*

Proof We need to prove that (RM1)–(RM3) hold for $\mathcal{L}'$, that $\emptyset \neq \mathcal{L}' \subseteq \mathcal{A}' \rightarrow \mathcal{G}'$, and that $\mathcal{L}'$ and $\mathcal{A}'$ do not deadlock.

Ad (RM1) Pick any $\alpha \in \mathrm{clo}\mathcal{L}'$ and $\gamma \in \Gamma'$, $s \in \mathrm{pfx}\,\alpha$ 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 $\gamma = f(\mathrm{p}_\Sigma s) \cup \{0\}$. We conclude that $\alpha \in \mathcal{L}'$ and obtain that $\mathcal{L}'$ is closed.

Ad (RM2) Pick an arbitrary $s \in (\mathrm{pfx}\,\mathcal{L}') \cap ((\Sigma'\Gamma')^*)$ and an arbitrary $\sigma' \in \Sigma'$. Observe that $\gamma = f(\mathrm{p}_\Sigma t) \cup \{0\}$ for all $t\gamma \leq s$, $\gamma \in \Gamma'$. Now let $\gamma' := f((\mathrm{p}_\Sigma t)\sigma')$ and $s' := s\sigma'\gamma'$. Thus, we again have $\gamma = f(\mathrm{p}_\Sigma t) \cup \{0\}$, but now for all $t\gamma \leq s'$, $\gamma \in \Gamma'$. Repeating this procedure, we obtain an ω-word $\alpha \in (\Sigma'\Gamma')^\omega$, $s\sigma' < \alpha$, such that $\gamma = f(\mathrm{p}_\Sigma t) \cup \{0\}$ for all $t\gamma < \alpha$, $\gamma \in \Gamma'$. In particular, we have $\alpha \in \mathcal{L}'$ and, hence $s\sigma' \in \mathrm{pfx}\,\mathcal{L}'$. Since the choice of $\sigma' \in \Sigma'$ was arbitrary, this implies (RM2).

Ad (RM3) 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 definition of $\mathcal{L}'$ this implies $\gamma = f(\mathrm{p}_\Sigma s) \cup \{0\} = \gamma'$ and we conclude (RM3).

Ad $\emptyset \neq \mathcal{L}'$ Pick an arbitrary 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 $\alpha \in \mathcal{L}'$.

Ad $\mathcal{L}' \subseteq \mathcal{A}' \rightarrow \mathcal{G}'$ We prove $\mathcal{A}' \cap \mathcal{L}' \subseteq \mathcal{G}'$. Pick any $\alpha \in \mathcal{A}' \cap \mathcal{L}'$. From $\alpha \in \mathcal{A}'$ and Eq. 34 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). We apply the projection p_Σ to $\alpha \in \mathcal{A}' \subseteq 0(\Gamma'\Sigma)^\omega$ and refer to the alternating structure to 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 $\mathrm{p}_\Sigma\alpha \in \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{G}$. Thus, we have that $\mathrm{p}_{\Sigma'}\alpha = 0\mathrm{p}_\Sigma\alpha \in 0\mathcal{G}$, which establishes $\alpha \in \mathcal{G}'$ by Eq. 33. Finally, we observe $\mathcal{L}' \subseteq \mathcal{A}' \rightarrow \mathcal{G}'$ from $\mathcal{A}' \cap \mathcal{L}' \subseteq \mathcal{G}'$.

Ad $\mathcal{A}'$ and $\mathcal{L}'$ not to deadlock Pick any $s \in (\mathrm{pfx}\,\mathcal{A}') \cap (\mathrm{pfx}\,\mathcal{L}')$. We distinguish two cases. For case (a), we 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 case (b), we have $s \in 0((\Gamma'\Sigma')^*)$. From $s \in (\mathrm{pfx}\,\mathcal{A}')$ and Eq. 34 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 apply the projection p_Σ to s 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 $\mathrm{p}_\Sigma s \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L})$. Since $\mathcal{A}$ and $\mathcal{L}$ are non-conflicting, we can choose $\sigma \in \Sigma$ such that $(\mathrm{p}_\Sigma s)\sigma \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L})$. In particular, $(\mathrm{p}_\Sigma s)\sigma \in \mathrm{pfx}\,\mathcal{L}$ implies 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 case (b).

Regarding regularity, consider a deterministic automaton realisation A of the topologically closed behaviour $\mathcal{L}$. By Eq. 54, ω-words $\beta \in \mathcal{L}'$ are constructed by inserting control

patterns in ω-words $\alpha \in \mathcal{L}$ after every single symbol. This construction can be realised by adding a marker-bit to the state set of A, effectively doubling the state count, and by extending the transition relation in order to track whether or not a control pattern is to be inserted next. Here, the control pattern to be inserted is identified as the set of enabled events in the corresponding state of A. In particular, the resulting automaton has a finite state count. □

Proof of Corollary 1 Assume that $\mathsf{SCT}[\Sigma,\ \Sigma_{\mathrm{uc}}, \mathcal{A},\ \mathcal{G}]$ has a (ω-regular) solution $\mathcal{L}$. We then refer to Proposition 6 to obtain a (ω-regular) solution of $\mathrm{RS}[UY\mathcal{A}'\mathcal{G}']$. Conversely, assume that $\mathrm{RS}[UY\mathcal{A}'\mathcal{G}']$ has a solution $\mathcal{L}'$. We then refer to Theorem 2 to obtain a (ω-regular) solution of $\mathsf{SCT}[\Sigma,\ \Sigma_{\mathrm{uc}}, \mathcal{A},\ \mathcal{G}]$. □

D Supervisory control via reactive synthesis – input-output behaviours

The following technical proposition summarises relevant properties obtained by our construction of $\mathcal{L}'$ in Eq. 36.

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. 36.

If $\mathcal{L}$ *satisfies (RM1) and (RM2) and if* pfx $\mathcal{L}$ *and* pfx $\mathcal{A}$ *do not deadlock, then* $\mathcal{L}'$ *satisfies (SC1) and (SC2).*

Moreover, $\mathcal{L}' \cap (UY)^\omega = \mathcal{L}$, (pfx $\mathcal{L}'$) $\cap$ (pfx$((UY)^*)$) = pfx $\mathcal{L}$. *Finally,* pfx $\mathcal{L}'$ *and* pfx $\mathcal{A}$ *do not deadlock.*

Proof We prove the individual claims suggested by the proposition.

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

$$\text{pfx } \mathcal{L}' \ = \ (\text{pfx } \mathcal{L}) \ \cup \ ((\text{pfx } \mathcal{L})\Sigma_{\mathrm{uc}}^*) \tag{55}$$

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 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 pfx $\mathcal{L}$ and, hence, infinitely many in (pfx $\mathcal{L}$)Σ_{uc}^*. In particular, there is a longest prefix $s < \alpha$ with $s \in$ pfx $\mathcal{L}$). Therefore, we have $\alpha \in s\Sigma_{\mathrm{uc}}^\omega \subseteq$ (pfx $\mathcal{L}$)$(\Sigma_{\mathrm{uc}}^\omega)$. In both cases, we have established $\alpha \in \mathcal{L}'$. Hence, $\mathcal{L}'$ is topologically closed.

Ad (SC2) For controllability, pick any $s \in$ pfx $\mathcal{L}'$ and $\sigma \in \Sigma_{\mathrm{uc}}$, to obtain $s\sigma \in$ pfx $\mathcal{L}'$ by Eq. 55.

Ad $\mathcal{L}' \cap (UY)^\omega = \mathcal{L}$ This equality is immediate from the fact that for every $\alpha \in \mathcal{L}' - \mathcal{L}$ we have $\alpha \in$ (pfx $\mathcal{L}$)$\Sigma_{\mathrm{uc}}^\omega$ and, hence, $\alpha \notin (UY)^\omega$.

Ad (pfx $\mathcal{L}'$) $\cap$ (pfx$((UY)^*)$) = pfx $\mathcal{L}$ We first show that $\mathcal{L}'$ and $(UY)^\omega$ are non-conflicting. Pick any $s \in$ (pfx $\mathcal{L}'$) $\cap$ (pfx$((UY)^\omega)$). If $s \in (UY)^*$, we refer to (SC2) and extend s by any $\sigma \in U = \Sigma_{\mathrm{uc}}$ to obtain $s\sigma \in$ 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 (55) for $t \in$ pfx $\mathcal{L}$, and to (RM2) for $s = tu \in$ pfx $\mathcal{L}$. Thus, we can extend s by $\sigma \in Y$ such that $s\sigma = tu\sigma \in$ pfx $\mathcal{L} \subseteq$ pfx $\mathcal{L}'$. In both cases, we have established $s\sigma \in$ (pfx $\mathcal{L}'$) $\cap$ (pfx$((UY)^\omega)$). Since both languages are topologically closed, this concludes the proof of non-conflictingness of $\mathcal{L}'$ and

$(UY)^\omega$, which in turn implies $(\text{pfx}\,\mathcal{L}') \cap (\text{pfx}((UY)^*)) = (\text{pfx}\,\mathcal{L}') \cap (\text{pfx}((UY)^\omega)) = \text{pfx}(\mathcal{L}' \cap ((UY)^\omega)) = \text{pfx}\,\mathcal{L}$.

Ad $\mathcal{A}$ and $\mathcal{L}'$ not to deadlock Note that $(\text{pfx}\,\mathcal{A}) \cap (\text{pfx}\,\mathcal{L}') = (\text{pfx}\,\mathcal{A}) \cap (\text{pfx}\,\mathcal{L}') \cap (\text{pfx}((UY)^\omega)) = (\text{pfx}\,\mathcal{A}) \cap (\text{pfx}\,\mathcal{L})$; 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 3 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 pfx $\mathcal{L}'$ and 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 (36), all operations retain regularity. □

Proof of Corollary 2 Assume that $\mathsf{SCT}[\Sigma,\ \Sigma_{\text{uc}},\ \mathcal{A},\ \mathcal{G}]$ has a (ω-regular) solution. We then refer to Theorem 1 to obtain a (ω-regular) solution of $\mathsf{RS}[U,\ Y,\ \mathcal{A},\ \mathcal{G}]$. Conversely, assume that $\mathsf{RS}[U,\ Y,\ \mathcal{A},\ \mathcal{G}]$ has a (ω-regular) solution. We then refer to Theorem 3 to obtain a (ω-regular) solution of $\mathsf{SCT}[\Sigma,\ \Sigma_{\text{uc}},\ \mathcal{A},\ \mathcal{G}]$.

□

Non-falsifiable assumptions and strong non-anticipation

Proposition 8 *Given an alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\text{uc}} \subsetneq \Sigma$, consider the reactive synthesis problem $\mathsf{RS}[U,\ Y,\ \mathcal{A},\ \mathcal{G}]$ with $U = \Sigma_{\text{uc}}$, $Y = \Sigma - \Sigma_{\text{uc}}$, and parameters $\mathcal{A}, \mathcal{G} \in (UY)^\omega$, both realised by deterministic Büchi automata.*

Furthermore let H be the game graph induced by $\mathcal{A}$ and $\mathcal{G}$ via M in Eq. 7 *with acceptance condition $\{T^0, T^1\}$. Then, the environment, player 0, has a winning strategy f^0 in the Büchi game (H, T^0) if and only if $\epsilon \in \text{cfx}_{\text{clo}\mathcal{A},\,Y}\mathcal{A}$.*

Proof We prove both implications separately. First, assume there exists a winning strategy f^0 for player 0 in the Büchi game (H, T^0). Then all plays compliant with f^0 have corresponding words $\alpha \in \mathcal{A}$. Now use f^0 to define the map $r : Y^+ \to U$ s.t. $r(\text{p}_Y s) := f^0(q)$ for $\delta(q_0, s) = q$. Then we can infer from Lemma 1 that the behaviour $\mathcal{L}$ associated with r satisfies (RM1)–(RM3), however, with Y the input symbols and U the output symbols. In particular, $\mathcal{L}$ is closed. Moreover, we have that $\mathcal{L} \cap (\text{clo}\mathcal{A}) \subseteq \mathcal{A}$ and that pfx $\mathcal{L}$ and pfx $\mathcal{A}$ do not deadlock. Now we can apply the transformation from Section 5.2.2 and, adapting the role of inputs and outputs accordingly, to obtain $\mathcal{L}' := \mathcal{L} \cup ((\text{pfx}\,\mathcal{L})((\Sigma - \Sigma_{\text{uc}})^\omega))$. Here, we refer to Proposition 7 to obtain that $\mathcal{L}'$ satisfies (SC1) and (SC2) with $Y = \Sigma - \Sigma_{\text{uc}}$ in the role of the uncontrollable events. In particular, we have that $\mathcal{L}'$ is closed and that $\mathcal{L}' \cap (\text{clo}\,\mathcal{A}) \subseteq \mathcal{A}$. Using $\mathcal{V} := \mathcal{L}' \cap (\text{clo}\,\mathcal{A})$, we have (i) $\mathcal{V} = \text{clo}\mathcal{V} = (\text{clo}\mathcal{V}) \cap (\text{clo}\mathcal{A})$ and (ii) $((\text{pfx}\,\mathcal{V})Y) \cap (\text{pfx}\,\mathcal{A}) \subseteq \text{pfx}\,\mathcal{V}$. Referring to Definition 2 this implies $\epsilon \in \text{cfx}_{\text{clo}\mathcal{A},\,Y}(\mathcal{A})$ and hence proves the first implication.

For the converse implication, assume that $\epsilon \in \text{cfx}_{\text{clo}\mathcal{A},\,Y}(\mathcal{A})$ and consider any deterministic Büchi automaton $M_\mathcal{A} = \{Q, \Sigma, \{q_0\}, \delta, F_\mathcal{A}\}$ such that $\mathcal{A} = \text{L}^\omega_{\text{m}}(M_\mathcal{A})$ and $\text{pfx}(\mathcal{A}) = \text{L}^*(M)$. Without loss of generality, we assume that $M_\mathcal{A}$ does not deadlock. Hence, using the trivial acceptance condition $\mathcal{F} = Q$, we can realise clo$\mathcal{A}$ on the same transition structure and we denote

$$M' = (Q, \Sigma, \{q_0\}, \delta, \{Q, F_\mathcal{A}\}) \tag{56}$$

a realisation of the formal plant $\mathcal{A}' = \text{clo}\mathcal{A}$ and the formal guarantee $\mathcal{G}' = \mathcal{A}$ with uncontrollable events $\Sigma'_{\text{uc}} := \Sigma - \Sigma_{\text{uc}} = Y$ and controllable events $\Sigma_{\text{uc}} = U$. This matches the setting of our discussion of the algorithmic solution to the synthesis problem, Section 4.3. Here, we refer to the simplified fixed-point in Eq. 25 for topologically closed plants, which for our problem parameters amounts to

$$\mathsf{Win}(M') = \nu X \,.\, \mu Z \,.\, \mathsf{Pre}(Z) \cup (F_{\mathcal{A}} \cap \mathsf{Pre}(X)) \tag{57}$$

with inverse dynamics operator, Eq. 22,

$$\mathsf{Pre}(T) = \{\, q \in Q \mid \delta(q, U \cup Y) \cap T \neq \emptyset \;\text{ and }\; \delta(q, Y) \subseteq T \,\}\,. \tag{58}$$

Recall from our discussion in Section 4.3 that a string is in the controllability prefix if and only if it leads to a winning state; i.e., technically $s \in \text{cfx}_{\mathcal{A}', \Sigma'_{\text{uc}}}(\mathcal{G}')$ if and only if $\delta(q_0, s) \in \mathsf{Win}(M')$. In particular, we obtain that $q_0 \in \mathsf{Win}(M')$ from the prerequisite $\epsilon \in \text{cfx}_{\text{clo}\mathcal{A},\, Y}(\mathcal{A}) = \text{cfx}_{\mathcal{A}', \Sigma'_{\text{uc}}}(\mathcal{G}')$.

We now turn to an interpretation in the context of reactive synthesis and recall that the alternation of input symbols and output symbols induces a disjoint union composition of the state set and the transition relation. Technically, we refer to Eq. 7 and write

$$M'' = (Q^0 \cup Q^1, U \cup Y, \{q_0\}, \delta^0 \cup \delta^1, \{T^0, T^1\}) \tag{59}$$

with $Q^0 \dot\cup Q^1 = Q$, $\delta^0 \dot\cup \delta^1 = \delta$, $q_0 \in Q^0$, $\delta^0 \subseteq Q^0 \times U \times Q^1$, $\delta^1 \subseteq Q^1 \times Y \times Q^0$, $T^0 = F_{\mathcal{A}}$, and with an arbitrary dummy parameter $T^1 \subseteq Q$. Note that, at this stage, we are not interested in solving a reactive synthesis problem. Instead, we consider the Büchi game (H'', T^0) from the perspective of player 0, where H'' denotes the game graph associated with M''. It is well known that this game can be solved via the fixed-point

$$\mathsf{Win}^0 = \nu X_4 \,.\, \mu X_3 \,.\, \mathsf{Pre}^0(X_3) \cup (T^0 \cap \mathsf{Pre}^0(X_4)) \tag{60}$$

with the player-0-controllable prefix defined

$$\mathsf{Pre}^0(T) := \{\, q \in Q^0 \mid \delta^0(q, U) \cap T \neq \emptyset \,\} \;\cup\; \{\, q \in Q^1 \mid \delta^1(q, Y) \subseteq T \,\}\,; \tag{61}$$

see also Eqs. 9 and 12, our argument in Remark 4 and the provided references (i.e., Maler et al. 1995; Zielonka 1998). In particular, there exists a winning strategy for player 0 if and only if $q_0 \in \mathsf{Win}^0$. We now observe (a) that the player-0-controllable prefix (61) matches the inverse dynamics operator (58), i.e., we have $\mathsf{Pre}^0(T) = \mathsf{Pre}(T)$ for any $T \subseteq Q$; and (b) that the two fixed-points (57) and (60) coincide. This implies that $\mathsf{Win}(M') = \mathsf{Win}^0$. In particular, we obtain that $q_0 \in \mathsf{Win}^0$ from $q_0 \in \mathsf{Win}(M')$, and, hence, there exists a winning strategy f^0 for player 0 in (H'', T^0).

Recall that our entire argument is valid for any deterministic Büchi automaton realisation $M_{\mathcal{A}}$ of $\mathcal{A}$. Hence, our argument also applies to the specific case where we choose $M_{\mathcal{A}}$ to have the same state set and the same transition relation as M from Eq. 7 obtained from the actual problem parameters $\mathcal{A}$ and $\mathcal{G}$. We then have that $M'' = M$ and, hence, the associated game graph H'' matches H. Thus player 0 has a winning strategy f^0 in the Büchi game (H, T^0). □

Proposition 9 *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 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* Eq. 38 *holds then $\mathcal{A}$ and $\mathcal{L}$ are non-conflicting.*

Proof Given any $s \in (\text{pfx}\ \mathcal{A}) \cap (\text{pfx}\ \mathcal{L})$, we need to show that $s \in \text{pfx}(\mathcal{A} \cap \mathcal{L})$. As a first step, we refer to the absence of deadlocks, Eq. 4, and optionally extend s by one symbol to obtain $s \leq s' \in (\text{pfx}\ \mathcal{A}) \cap (\text{pfx}\ \mathcal{L}) \cap (UY)^*$.

Referring to Eq. 38 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. $(\text{clo}\mathcal{A}) \cap (s'\Sigma^\omega)$ and (ii) $((\text{pfx}\ \mathcal{V})Y) \cap (\text{pfx}\ \mathcal{A}) \cap (s'\Sigma^*) \subseteq (\text{pfx}\ \mathcal{V})$. Since $(\text{clo}\mathcal{A}) \cap (s'\Sigma^\omega)$ is topologically closed, (i) implies that $\mathcal{V}$ is closed, too.

We now construct a strictly monotone sequence $(t_i)_{i\in\mathbb{N}}$ with $t_i < t_{i+1}$ and

$$s't_i \in (\text{pfx}\ \mathcal{V}) \cap (\text{pfx}\ \mathcal{L}) \cap (UY)^* . \tag{62}$$

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 $u_i \in U$ such that $s't_iu_i \in \text{pfx}\ \mathcal{V}$ to observe $s't_iu_i \in \text{pfx}\ \mathcal{L}$ by the free input (RM2). Likewise, we refer to the absence of deadlocks, Eq. 4, and choose $y_i \in Y$ such that $s't_iu_iy_i \in (\text{pfx}\ \mathcal{A}) \cap (\text{pfx}\ \mathcal{L})$. By property (ii) of $\mathcal{V}$, we obtain $s't_iu_iy_i \in (\text{pfx}\ \mathcal{V}) \cap (\text{pfx}\ \mathcal{L})$. Now $t_{i+1} := t_iu_iy_i$ satisfies the requirements to conclude the iterative construction of $(t_i)_{i\in\mathbb{N}}$.

Denote β the singleton limit of the sequence $(t_i)_{i\in\mathbb{N}}$. Then $s'\beta$ has infinitely many prefixes in pfx $\mathcal{V}$ and 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, $s \in \text{pfx}(\mathcal{A} \cap \mathcal{L})$. □

Proof of Theorem 4 As in the proof of Theorem 3, 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 $(\text{pfx}\ \mathcal{A}) \cap (\text{pfx}\ \mathcal{L}') = (\text{pfx}\ \mathcal{A}) \cap (\text{pfx}\ \mathcal{L})$. For non-conflictingness, pick $s \in (\text{pfx}\ \mathcal{A}) \cap (\text{pfx}\ \mathcal{L}')$,

and refer to Proposition 9 for $s \in \text{pfx}(\mathcal{A} \cap \mathcal{L}) = \text{pfx}(\mathcal{A} \cap \mathcal{L}')$. Again, non-conflictingness and non-emptyness of $\mathcal{A}$ and $\mathcal{L}'$ imply a non-empty closed-loop behaviour. □

For a Proof of Corollary 3, substitute the reference to Theorem 3 by Theorem 4 in the Proof of Corollary 2.

References

Baier C, Moor T (2015) A hierarchical and modular control architecture for sequential behaviours. J Discret Event Dyn Syst 25:95–124

Barati M, St-Denis R (2015) Behavior composition meets supervisory control. In: 2015 IEEE International Conference on Systems, Man, and Cybernetics, pp 115–120. https://doi.org/10.1109/SMC.2015.33

Barveau M, Kabanza F, St-Denis R (1998) A method for the synthesis of controllers to handle safety, liveness, and real-time constraints. IEEE Trans Autom Control 43(11):1543–1559. ISSN 0018-9286. https://doi.org/10.1109/9.728871

Bloem R, Jobstmann B, Piterman N, Pnueli A, Sahar Y (2012) Synthesis of reactive(1) designs. J Comput Syst Sci 78(3):911–938

Bloem R, Ehlers R, Jacobs S, Könighofer R (2014) How to handle assumptions in synthesis. In: SYNT'14 Vienna, Austria, pp 34–50

Bloem R, Ehlers R, Könighofer R (2015) Cooperative reactive synthesis. In: ATVA Shanghai, China, pp 394–410

Bourdon E, Lawford M, Wonham WM (2005) Robust nonblocking supervisory control of discrete-event systems. IEEE Trans Autom Control 50:2015–2021

Bradfield J, Stirling C (2006) Modal mu-calculi. In: The Handbook of Modal Logic. Elsevier, pp 721–756

Brenguier R, Raskin J-F, Sankur O (2017) Assume-admissible synthesis. Acta Inform 54(1):41–83

Büchi JR, Landweber LH (1969) Solving sequential conditions by finite-state strategies. Trans Amer Math Soc 138:367–378
Cai K, Zhang R, Wonham WM (2015) Relative observability of discrete-event systems and its supremal sublanguages. IEEE Trans Autom Control 60(3):659–670
Chatterjee K, Henzinger TA (2007) Assume-guarantee synthesis. In: TACAS 2007, Braga, Portugal, pp 261–275
Chatterjee K, Horn F, Löding C (2010) Obliging games. In: CONCUR'10, Paris, France, pp 284–296
Chatterjee K, Henzinger TA (2012) A survey of stochastic ω,-regular games. J Comput Syst Sci 78(2):394–413. https://doi.org/10.1016/j.jcss.2011.05.002
Church A (1957) Applications of recursive arithmetic to the problem of circuit synthesis. Summ Summer Inst Symbol Log I:3–50
Cieslak R, Desclaux C, Fawaz AS, Varaiya P (1988) Supervisory control of discrete-event processes with partial observations. IEEE Trans Autom Control 33(3):249–260
Cury J, Krogh B (1999) Robustness of supervisors for discrete-event systems. IEEE Trans Autom Control 44:376–379
de Alfaro L, Henzinger TA (2000) Concurrent omega-regular games In: Proceedings Fifteenth Annual, IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332), pp 141–154
de Alfaro L, Henzinger TA, Kupferman O (2007) Concurrent reachability games. Theor Compute Sci 386(3):188–217
de Querioz MH, Cury JER (2000) Modular supervisory control of large scale discrete event systems. WODES
Ehlers R, Lafortune S, Tripakis S, Vardi MY (2017) Supervisory control and reactive synthesis: a comparative introduction. Discret Event Dyn Syst 27(2):209–260
Emerson E, Jutla C (1991) Tree automata, mu-calculus and determinacy. In: FOCS'91, pp 368–377
Felli P, Yadav N, Sardina S (2017) Supervisory control for behavior composition. IEEE Trans Autom Control 62(2):986–991. ISSN 0018-9286. https://doi.org/10.1109/TAC.2016.2570748
Finkbeiner B (2016) Synthesis of reactive systems. Technical report, Universität des Saarlandes
Fisman D, Kupferman O, Lustig Y (2010) Rational synthesis. In: TACAS'10, pp 190–204
Grädel E, Thomas W, Wilke T (eds.) (2002) Automata Logics, and Infinite Games, volume 2500 of LNCS. Springer, Berlin
Gurevich Y, Harrington L (1982) Trees, automata, and games. In: STOC '82,San Francisco, pp 60–65
Hopcroft JE, Ullman JD (1979) Introduction to automata Theory, languages and computation. Addison-Wesley, Reading
Kučera V (2011) A method to teach the parameterization of all stabilizing controllers. IFAC Proc Vol 44(1):6355–6360. 18th IFAC World Congress
Kupferman O, Vardi MY (2000) Synthesis with incomplete information. In: Advances in Temporal Logic, pp 109–127
Lin F, Wonham WM (1988) On observability of discrete-event systems. Inf Sci 44:173–198
Majumdar R, Piterman N, Schmuck A-K (2019) Environmentally-friendly GR(1) synthesis. In: TACAS'19. Springer, pp 229–246
Maler O, Pnueli A, Sifakis J (1995) On the synthesis of discrete controllers for timed systems, pp 229–242
Moor T, Schmidt K, Wittmann T (2011) Abstraction-based control for not necessarily closed behaviours. Proceedings of the 18th IFAC World Congress, pp 6988–6993
Moor T, Baier C, Yoo T-S, Lin F, Lafortune S (2012) On the computation of supremal sublanguages relevant to supervisory control 45(29):175–180. WODES 2012
Moor T (2016) A discussion of fault-tolerant supervisory control in terms of formal languages. Annu Rev Control 41:159–169
Moor T (2017) Supervisory control on non-terminating processes: An interpretation of liveness properties. Technical report, Lehrstuhl für Regelungstechnik, Friedrich-Alexander Universität Erlangen-Nürnberg
Paoli A, Sartini M, Lafortune S (2011) Active fault tolerant control of discrete event systems using online diagnostics. Automatica 47(4):639–649
Pnueli A (1977) The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science. IEEE Computer Society Press, pp 46–57
Pnueli A, Rosner R (1989) On the synthesis of a reactive module. In: SIGPLAN-SIGACT. ACM, New York, pp 179–190
Rabin MO (1972) Automata on infinite objects and church's problem. American Mathematical Society, Boston
Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25:206–230
Ramadge PJ (1989a) Some tractable supervisory control problems for discrete-event systems modeled by büchi automata. IEEE Trans Autom Control 34:10–19

Ramadge PJ, Wonham WM (1989b) Modular control of discrete event systems Maths. of Control. Signals Syst 1:1:13–30
Rudie K, Wonham WM (1992) Think globally, act locally: decentralized supervisory control. IEEE Trans Autom Control 37:11:1692–1708
Safra S (1988) On the complexity of omega-automata. In: FOCS'88, pp 319–327
Schmidt K, Moor T, Perk S (2008) Nonblocking hierarchical control of decentralized discrete event systems. IEEE Trans Autom Control 53(10):2252–2265
Schmuck A-K, Moor T, Majumdar R (2018) On the relation between reactive synthesis and supervisory control for input/output behaviours. WODES
Thistle JG (1995) On control of systems modelled as deterministic rabin automata. Discret Event Dyn Syst 5(4):357–381
Thistle JG, Wonham WM (1992) Control of omega-automata, church's problem, and the emptiness problem for tree omega-automata. Proceedings of the 5th Workshop on Computer Science Logic, pp 367–382
Thistle JG, Wonham WM (1994a) Supervision of infinite behavior of discrete event systems. SIAM J Control Optim 32:1098–1113
Thistle JG, Wonham WM (1994b) Control of infinite behavior of finite automata. SIAM J Control Optim 32:1075–1097
Thistle JG, Lamouchi HM (2009) Effective control synthesis for partially observed discrete-event systems. SIAM J Control Optim 48:1858–1887
Thomas W (1990) Automata on infinite objects. In Handbook of Theoretical Computer Science (Vol. B). MIT Press, pp 133–191
Thomas W (1995) On the synthesis of strategies in infinite games. In: STACS'95 Munich, Germany, pp 1–13
Vardi MY, Wolper P (1986) Automata-theoretic techniques for modal logics of programs. J Comput Syst Sci 32:183–221
Wen Q, Kumar R, Huang J, Liu H (2008) A framework for fault-tolerant control for discrete event systems. IEEE Trans Autom Control 53:1839–1849
Willems JC (1991) Paradigms and puzzles in the theory of dynamic systems. IEEE Trans Autom Control 36:258–294
Wong KC, Wonham WM (1996) Hierarchical control of discrete-event systems. Discret Event Dyn Syst Theory Appl 6(3):241–273
Yin X, Lafortune S (2016) Synthesis of maximally permissive supervisors for partially-observed discrete-event systems. IEEE Trans Automat Contr 61(5):1239–1254
Yin X, Lafortune S (2017) Synthesis of maximally-permissive supervisors for the range control problem. IEEE Trans Autom Control 62(8):3914–3929
Zhang R, Cai K (2018) Supervisor localization of discrete-event systems with infinite behavior. WODES, pp 361–366
Zhong H, Wonham WM (1990) On the consistency of hierarchical supervision in discrete-event systems. IEEE Trans Autom Control 35:1125–1134
Zielonka W (1998) Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor Comput Sci 200(1-2):135–183

Anne-Kathrin Schmuck received the Dipl.-Ing. (M.Sc) degree in engineering cybernetics from OvGU Magdeburg, Germany, in 2009 and the Dr.-Ing. (Ph.D.) degree in electrical engineering from TU Berlin, Germany, in 2015. She is currently a postdoctoral researcher at the MPI-SWS in Kaiserslautern, Germany. From 2010 to 2015 she was a research assistant in the Control Systems Group at TU Berlin, Germany. Her current research interests include abstraction based controller synthesis, reactive synthesis, supervisory control theory and hierarchical control.

Thomas Moor received his PhD degree (Dr.-Ing.) in 1999 from the University of the Federal Armed Forces Hamburg. From 2000 to 2003 he was a research fellow with the Research School of Information Sciences and Engineering at the Australian National University. Since 2003, he holds a professorship at the Lehrstuhl für Regelungstechnik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. His research interests include the control of discrete-event systems and hybrid systems, hierarchical and/or modular control systems, control system abstraction and fault-tolerant control. He is maintainer and principle developer of the discrete-event systems software library libFAUDES, with a particular focus on supervisory control in an industrial application context.

Rupak Majumdar is a Scientific Director at the Max Planck Institute for Software Systems. His research interests are in the verification and control of reactive, real-time, hybrid, and probabilistic systems, software verification and programming languages, logic, and automata theory. He received the B.Tech. degree in Computer Science from the Indian Institute of Technology at Kanpur and the Ph.D. degree in Computer Science from the University of California at Berkeley.

Affiliations

Anne-Kathrin Schmuck[1] · Thomas Moor[2] · Rupak Majumdar[1]

Thomas Moor
lrt@fau.de

Rupak Majumdar
rupak@mpi-sws.org

1 Max Planck Institute for Software Systems, Kaiserslautern, Germany

2 Lehrstuhl für Regelungstechnik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Nürnberg, Germany