On the Relation between Reactive Synthesis and Supervisory Control of Input/Output Behaviours

Anne-Kathrin Schmuck [*] **Thomas Moor** [**] **Rupak Majumdar** [*]

[*] *Max Planck Institute for Software Systems, Kaiserlautern, Germany,* {*akschmuck,rupak*}*@mpi-sws.org.*
[**] *Lehrstuhl für Regelungstehnik, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany, lrt@fau.de*

Abstract: *Reactive synthesis* (RS) and *supervisory control theory* (SCT) both provide a design methodology for digital systems. *RS* takes a computer science perspective and seeks to synthesise a system that interacts with its environment in computation cycles and which, doing so, satisfies a prescribed specification. *SCT* takes a control theoretic perspective and seeks to synthesise a controller that – in closed-loop configuration with a plant – enforces a prescribed specification, where all dynamics are driven by discrete events. While both synthesis techniques seem superficially very similar, their technical details differ significantly. We provide a formal comparison allowing us to identify conditions under which one can solve one synthesis problem via the other one and we discuss how the resulting solutions compare. To facilitate this comparison, we give a unified introduction to RS and SCT and derive formal problem statements and a characterisation of their solutions in terms of ω-languages. As recent contributions to the two fields focus on different aspects of the respective problem, our translational theorems can be used to guide the application of algorithmic techniques from one field in the other.

1. INTRODUCTION

Reactive synthesis (RS) is a branch of computer science and addresses the automated synthesis of digital systems that dynamically interact with their environment in a feedback configuration s.t. some prescribed specification is met. The problem of RS was first proposed by Church (1957) with solutions provided by Büchi and Landweber (1969), and Rabin (1972). It is since then an active field of research, addressing temporal logic specifications (e.g. Gurevich and Harrington (1982); Pnueli and Rosner (1989); Emerson and Jutla (1991)), partial observation (e.g. Kupferman and Vardi (2000)), and, most relevant for the present report, environment assumptions (see Bloem et al. (2014) and Brenguier et al. (2017) for an overview). For a comprehensive introduction to the field see e.g. Thomas (1995); Finkbeiner (2016).

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

Both design methodologies seek to synthesize a causal feedback map that operates on a finite alphabet and that satisfies a formal specification. When used to synthesize solutions for particular instances of a given synthesis problem, both techniques and their obtained solutions seem superficially very similar. 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 discuss how the resulting solutions compare. As recent contributions to the two fields of research focus on different aspects of the respective problem, our translational theorems can be used to guide the application of other algorithmic techniques from one field in the other.

Scope Regarding RS, our study considers a variant that explicitly addresses assumptions on the environment behaviour, which correspond to the prescribed plant behaviour in SCT. RS typically uses specifications given in linear temporal logic (LTL); it is well understood how such formula can be translated into ω-regular languages (Safra (1988); Vardi and Wolper (1986)). For ease of presentation, we consider both the assumption and the specification to be given either abstractly as ω-languages, i.e., formal languages of infinite words, or, for illustration purposes, as *deterministic* Büchi automata.

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

Contribution Within this perspective of our choice, our contribution is twofold: (I) We show that one can solve the considered RS problem using SCT to obtain a reactive module which will not falsify the assumptions on the environment. (II) We show that one can solve the considered synthesis problem from SCT using RS if the given assumptions are not falsifiable.

Intuitively, the considered RS problem is formalised by an implication style logic formula, i.e., if the assumptions are satisfied, then the specification must be enforced. 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 fulfill their liveness properties eventually. Due to this additional property of solutions, our transformation in result (I) achieves reactive modules which do not falsify the assumption. For the reverse transformation (result (II)), however, we have to assume that the given assumptions are not falsifiable by a reactive module to ensure that RS returns solutions which are non-conflicting and hence a solution to the initial SCT synthesis problem.

Both synthesis algorithms are formalized as fixed-points in μ-calculus. The RS algorithm uses a 3-nested fixed-point while in SCT synthesis amounts to a 4-nested fixed-point. Result (I) clarifies that this additional iteration indeed generates an additional property on the solution. Regarding result (II), we may expect computational benefits as a trade-off when imposing additional conditions for the synthesis problem in SCT.

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

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

Observing page constraints, we provide supporting technical propositions, proofs of our main results and many illustrative examples in the technical report[1] by Schmuck et al. (2017).

[1] Available from the first authors' home page.

2. PRELIMINARIES[2]

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 $s \le t$ and $s < t$ if s is a prefix of t or a strict prefix of t, respectively. All prefixes of a word $t \in \Sigma^\infty$ are 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 $\mathcal{L} = (\mathrm{clo}\, \mathcal{L}) \cap \mathcal{M}$.

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 $\delta(q, \sigma) := \{\, q' \mid (q, \sigma, q') \in \delta \,\}$, and with the common inductive extension to a word-valued second argument $s \in \Sigma^*$. If $|Q_\mathrm{o}| \le 1$ and $|\delta(q, s)| \le 1$ 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 $\delta(q, s) \ne \emptyset$, respectively. We define $L = \mathrm{L}^*(M) := \{\, s \in \Sigma^* \mid \delta(Q_\mathrm{o}, s) \ne \emptyset \,\}$ 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 Generated languages can be restricted by *acceptance conditions*. Given a set of *final states* $F \subseteq Q$ and the extended automaton tuple $M = (Q, \Sigma, \delta, Q_\mathrm{o}, F)$, the *accepted* $*$-language is defined $\mathrm{L}^*_\mathrm{m}(M) := \{\, s \in \Sigma^* \mid \delta(Q_\mathrm{o}, s) \cap F \ne \emptyset \,\}$. *Regular $*$-languages* are those that are accepted by a finite automaton. For ω-languages, we consider the *Büchi* and the *generalized Büchi* acceptance condition, given by a set $\mathcal{F} \subseteq Q$ and a family of sets $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$ with $F_i \subseteq Q$, respectively. An automaton $M = (Q, \Sigma, \delta, Q_\mathrm{o}, \mathcal{F})$ with (generalized) Büchi acceptance condition is referred to as a *(generalized) Büchi automaton*; every generalized Büchi automaton with $\mathcal{F} = \{F\}$ (i.e., $k = 1$) can be redefined to a Büchi automaton with final state set $\mathcal{F} = F$. A run π of M is an infinite sequence of states $q_1 q_2 q_3 \cdots \in Q^\omega$ and corresponds to the ω-word $\alpha = \sigma_1 \sigma_2 \sigma_3 \cdots \in \Sigma^\omega$ if $q_1 \in Q_\mathrm{o}$ and $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}$. The run π is accepted by a (generalized) Büchi automaton, if $(\mathrm{Inf}\, \pi) \cap F_i \ne \emptyset$ for all $i \in \{1, \ldots, k\}$, where $\mathrm{Inf}\, \pi$ denotes the set of states that occur infinitely often in π. The accepted ω-language $\mathrm{L}^\omega_\mathrm{m}(M)$ consists of all words $\alpha \in \Sigma^\omega$ for which there exists a corresponding accepting run over M. For deterministic automata, we have $\mathrm{L}^\omega_\mathrm{m}(M) = \lim \mathrm{L}^*_\mathrm{m}(M)$. The class of ω-languages that is accepted by a finite Büchi automaton is referred to as the ω-regular languages; the class of ω-languages that is accepted by a *deterministic* finite Büchi automaton is a strict subset of the former.

[2] A general introduction to these topics can be found in e.g. Hopcroft and Ullman (1979); Thomas (1990); Bradfield and Stirling (2006).

Fixpoint Calculus We utilise 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 $\mu P.f(P) = \cup\{ P_i \,|\, i \in \mathbb{N} \}$, and $P_1 := Q$, $P_{i+1} := P_i \cap f(P_i)$, with $\nu P.f(P) = \cap\{ P_i \,|\, i \in \mathbb{N} \}$. 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, hence, fixed-point formulae can be nested.

3. REACTIVE SYNTHESIS

Reactive Modules A *reactive module* is a device that reads 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, $r : U^+ \to Y$. 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:

$$\mathcal{L} := \{ \alpha \in (UY)^\omega \,| \\ \forall\, s \in (U \cup Y)^*,\, y \in Y \,.\, sy < \alpha \;\Rightarrow\; y = r(\mathrm{p}_U(s)) \} \,. \quad (1)$$

Note that, by construction, $\mathcal{L}$ is topologically closed. Moreover, if r is realised as a finite automaton, then $\mathcal{L}$ is ω-regular.

For our subsequent discussion we eliminate the explicit reference to the system $r : U^+ \to Y$ by utilizing a more direct characterisation of those languages $\mathcal{L}$ that qualify for a representation by (1). For this purpose, we adapt the notion of *input-output systems* from *behavioural systems theory*, Willems (1991), for the special case of topologically closed languages and input/output behaviours.

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

(i) *U is a locally free input for $\mathcal{L}$* if for all $s \in \mathrm{pfx}\,\mathcal{L}$ and $u', u'' \in U$ we have that $su' \in \mathrm{pfx}\,\mathcal{L}$ implies $su'' \in \mathrm{pfx}\,\mathcal{L}$;

(ii) the *output locally processes the input* if for all $s \in \mathrm{pfx}\,\mathcal{L}$ and $y', y'' \in Y$ with sy', $s''y \in \mathrm{pfx}\,\mathcal{L}$ we have $y' = y''$. □

The above notions enable the following characterisation of behaviours associated with a reactive module.

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

(RM1) $\mathcal{L}$ is topologically closed,
(RM2) the input is locally free, and
(RM3) the output 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

$$\{ y \in Y \mid \exists\, s \in (UY)^* U \;.\; \mathrm{p}_U\, s = v \wedge sy \in \mathrm{pfx}\,\mathcal{L} \} \qquad (2)$$

for all $v \in U^+$. □

Problem Statement The problem commonly referred to as *reactive synthesis* is about the systematic design of a reactive module, such that its behaviour satisfies an upper-bound specification[3] $\mathcal{G} \subseteq (UY)^\omega$. The crucial point here is that $\mathcal{G}$ is ω-regular, but in general fails to be topologically closed.

In the basic setting of reactive synthesis, the behaviour of the environment is unrestricted which makes the synthesis problem hard to solve. However, in many applications one has prior knowledge about the behaviour of the environment. This can be incorporated into the synthesis problem by defining a language $\mathcal{A} \subseteq (UY)^\omega$, which models the assumptions on the generation of inputs to the system. If environment assumptions are given, the reactive synthesis problem amounts to designing a reactive module r whose behaviour $\mathcal{L}$ satisfies the specification $\mathcal{G}$ whenever the assumptions $\mathcal{A}$ are fulfilled. To ensure that the designed reactive module r can interact with the environment in non-terminating computation cycles, it is required that the local interaction of environment and system does not deadlock, i.e.,

$$\forall\, s \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L}) \,.\, \exists\, \sigma \in U \cup Y : \\ s\sigma \in (\mathrm{pfx}\,\mathcal{A}) \cap (\mathrm{pfx}\,\mathcal{L}) \,. \quad (3)$$

These two requirements are formalized in the following formal problem statement.

Problem 1. (Reactive Synthesis). Given two non-empty finite sets U and Y, $U \cap Y = \emptyset$, and the ω-languages $\mathcal{G} \subseteq (UY)^\omega$ and $\mathcal{A} \subseteq (UY)^\omega$, either construct a system $r : U^+ \to Y$ such that the associated behaviour $\mathcal{L}$ does not deadlock with $\mathcal{A}$, see Eq. (3), and such that

$$\emptyset \neq \mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}\,, \qquad (4)$$

or verify that no such system exists. □

Note that, for $\mathcal{A} = \emptyset$, the upper bound $\mathcal{A} \to \mathcal{G}$ degenerates and the specification becomes $\mathcal{L} \subseteq (UY)^\omega$. Thus, in our discussion of the above problem we may whenever convenient assume that $\mathcal{A} \neq \emptyset$ and, likewise, $\mathcal{G} \neq \Sigma^\omega$. Moreover, we have that $\mathcal{A} \to (\mathcal{G} \cap \mathcal{A}) = (\Sigma^\omega - \mathcal{A}) \cup (\mathcal{G} \cap \mathcal{A}) = \mathcal{A} \to \mathcal{G}$, and, hence, we can restrict our discussion without loss of generality to the case $\mathcal{G} \subseteq \mathcal{A}$.

With Lemma 1, the problem of reactive synthesis, Problem 1, amounts to the construction of a non-empty subset $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$ that satisfies (RM1) – (RM3), or to the verification that no such subset exists. Henceforth, we may refer to a qualifying behaviour $\mathcal{L}$ as a *solution* of Problem 1.

Algorithmic Solution The interaction of the system and its environment can be viewed as a turn-based two player game: in every round the environment player selects an arbitrary input $u \in U$ and the system selects the output $y \in Y$ according to r. It was shown by Gurevich and Harrington (1982); Pnueli and Rosner (1989) that for ω-regular specifications there exists a winning strategy for the system player in this game if and only if Problem 1 has a ω-regular solution $\mathcal{L}$. Based on this result, Problem 1 can be solved 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 r. For a concise presentation of this construction, we consider the special case in which both $\mathcal{G}$ and $\mathcal{A}$ are realised as deterministic Büchi automata and in which we assume that $\mathcal{G} \subseteq \mathcal{A}$. In this case there is a direct and simple procedure to solve Problem 1 which we briefly recall.

[3] Note that specifications given in Linear Temporal Logic (LTL) effectively amount to such upper bound specifications.

Consider the generalized Büchi automaton

$$M = (Q^0 \cup Q^1, U \cup Y, q_0, \gamma^0 \cup \gamma^1, \{T^0, T^1\}) \qquad (5)$$

s.t. $q_0 \in Q^0$, $\gamma^0 \subseteq Q^0 \times U \times Q^1$ and $\gamma^1 \subseteq Q^1 \times Y \times Q^0$. Furthermore, M exhibits no deadlocks, generates the $*$-language $\mathrm{pfx}(\mathcal{A}) = \mathrm{L}^*(M)$ and accepts the ω-languages $\mathcal{A} = \mathrm{L}^\omega_\mathrm{m}(M^0)$ and $\mathcal{G} = \mathrm{L}^\omega_\mathrm{m}(M^1)$ where M^0 and M^1 refer to the simple Büchi automaton obtained from M by using the single winning state set T^0 and T^1, respectively. M directly defines a turn based deterministic two player game graph $H = (Q^0, Q^1, U, Y, \gamma^0, \gamma^1)$. Recalling Problem 1, a system winning strategy for this game must ensure that all plays on H that visit T^0 infinitely often, must also visit T^1 infinitely often, which can be formalized as a *deterministic parity game* with three colours. Such games can be solved using the three-nested fixed point

$$\mathsf{Win}^1(C) = \nu X_4 \,.\, \mu X_3 \,.\, \nu X_2 \; \cup_{k=2}^{4} \,(\, C_k \cap \mathsf{Pre}^1(X_k)\,)\,. \qquad (6)$$

where $C_2 = Q \setminus T^0$, $C_3 = T^0$, $C_4 = T^1$, and

$$\mathsf{Pre}^1(X) := \{\, v^0 \in V^0 \mid \forall\, u \in U \,.\, \delta^0(v^0, u) \in X \,\} \\ \cup \{\, v^1 \in V^1 \mid \exists\, y \in Y \,.\, \delta^1(v^1, y) \in X \,\}. \qquad (7)$$

If $q_0 \in \mathsf{Win}^1(C)$, the module solving Problem 1 can be extracted from the iterations over X_k in (6), which constitutes a special case of the construction presented in Bloem et al. (2012).

Remark 1. For the special case of a topologically closed assumption $\mathcal{A}$ 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 (6) simplifies to

$$\mathsf{Win}^1(C) = \nu X_4 . \mu X_3 . \mathsf{Pre}^1(X_3) \cup (F^1 \cap \mathsf{Pre}^1(X_4))\,. \qquad (8)$$

Note that (8) solves the Büchi game (H, T^1), which is obtained when solving the original version of Problem 1 where the environment behaviour is unconstrained and hence given by $\mathcal{A} = (UY)^\omega$ (see e.g. Maler et al. (1995); Zielonka (1998)).

4. SUPERVISORY CONTROL

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

$$f : \Sigma^* \to \Gamma\,, \quad \text{with} \quad \Gamma := \{\, \gamma \subseteq \Sigma \,|\, \Sigma_\mathrm{uc} \subseteq \gamma \,\}\,, \qquad (9)$$

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. From a reactive synthesis point of view, the supervisor is the system we seek to design and, for practical purposes, a realisation as a finite automaton is of a particular interest.

For the subsequent discussion, we define the *behaviour associated with the supervisor f* as the ω-language [4]

$$\mathcal{L} := \{\, \alpha \in \Sigma^\omega \mid \forall s \in \Sigma^*, \sigma \in \Sigma \,.\, s\sigma < \alpha \Rightarrow \sigma \in f(s) \,\}. \qquad (10)$$

The following lemma characterises languages that match the behaviour of some supervisor.

[4] The proposed representation of a supervisor f by the ω-language $\mathcal{L}$ does not account for supervisors which deadlock, 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.

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

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

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

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

$$f'(s) := \{\, \sigma \in \Sigma \,|\, s\sigma \in \mathrm{pfx}\,\mathcal{L} \,\} \cup \Sigma_\mathrm{uc} \qquad (11)$$

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

Problem Statement The problem commonly referred to as *supervisory controller synthesis* is about the systematic design of a supervisor, s.t. the resulting closed-loop system – established by the feedback composition of this supervisor with the plant – satisfies a given specification. Referring to the reactive synthesis perspective, this identifies the plant as the environment, formally represented as an ω-language $\mathcal{A} \subseteq \Sigma^\omega$.

Given a plant and a supervisor, the closed-loop configuration is defined to evolve on words that comply with both component behaviours. Technically, this amounts to the *local closed-loop behaviour* $K_\mathrm{loc} := (\mathrm{pfx}\,\mathcal{L}) \cap \mathrm{pfx}(\mathcal{A})$ and the *accepted closed-loop behaviour* $\mathcal{K} := \mathcal{L} \cap \mathcal{A}$. Regarding liveness of the closed-loop configuration, supervisory control commonly addresses not only deadlocks but also livelocks. The latter are characterised by finite sequences $s \in K_\mathrm{loc}$ from the local closed-loop behaviour that can be continued indefinitely within K_loc but any such infinite extension fails to satisfy the plant acceptance condition. Technically, we ask for a *non-blocking supervisor*, i.e., we require that both languages are *non-conflicting*:

$$(\mathrm{pfx}\,\mathcal{L}) \cap (\mathrm{pfx}\,\mathcal{A}) = \mathrm{pfx}(\mathcal{L} \cap \mathcal{A})\,. \qquad (12)$$

Consequently, we refer to $\mathcal{K} := \mathcal{L} \cap \mathcal{A}$ as the model of the closed-loop configuration and have $K_\mathrm{loc} = \mathrm{pfx}\,\mathcal{K}$.

For the purpose of our discussion, we observe that an upper-bound specification $\mathcal{K} \subseteq \mathcal{G}$ can be interpreted as a guarantee with the particular feature, that any supervisor enforcing this guarantee on the closed loop behaviour cannot invalidate the assumption $\mathcal{A}$, i.e., $\mathcal{K} \subseteq \mathcal{A}$. Hence, if we establish a supervisor such that the closed-loop behaviour $\mathcal{K}$ is non-empty and satisfies $\mathcal{K} \subseteq \mathcal{G}$, we trivially obtain $\mathcal{K} \subseteq \mathcal{A} \cap \mathcal{G} \subseteq \mathcal{A} \to \mathcal{G}$.

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

Problem 2. (Supervisory Controller Synthesis). Given an alphabet Σ with the non-empty set of uncontrollable events $\Sigma_\mathrm{uc} \subseteq \Sigma$, a plant $\mathcal{A} \subseteq \Sigma^\omega$ and an upper-bound specification $\mathcal{G} \subseteq \Sigma^\omega$, construct a supervisor with associated behaviour $\mathcal{L} \subseteq \Sigma^\omega$ that is non-blocking, see Eq. (12), and that satisfies

$$\emptyset \neq \mathcal{A} \cap \mathcal{L} \subseteq \mathcal{G}\,, \qquad (13)$$

or verify that no such supervisor exists. □

Referring to the behavioural characterisation of supervisors in Lemma 2, we identify a qualifying associated behaviour $\mathcal{L}$ with a *solution* to Problem 2. Note that the trivial case of $\Sigma = \Sigma_\mathrm{uc}$ implies $\mathcal{L} = \Sigma^\omega$ and the synthesis problem collapses to the verification of $\mathcal{A} \subseteq \mathcal{G}$.

Controllability Prefix To solve Problem 2, Ramadge (1989) characterises all closed-loop behaviours that can be achieved by non-blocking supervisory control for a given plant.

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

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

Since (i) is not retained under arbitrary union, a maximally permissive solution does not exist in general and this contrasts SCT for $*$-languages. To compute specific closed-loop behaviours solving Problem 2, we utilize the notion of the *controllability prefix* introduced by Thistle and Wonham (1994b).

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

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

Comparing the above conditions (i) and (ii) with Proposition 1, we see that $\mathcal{V}_s$ is a closed-loop behaviour that can be enforced by a non-blocking supervisor, if it "takes over to control the plant" after the word $s \in \mathrm{pfx}_{\mathcal{A}}(\mathcal{G})$ was generated by the plant. As $\mathcal{V}_s \subseteq \mathcal{G}$ this supervisor is able to enforce the guarantee $\mathcal{G}$. In particular, the synthesis problem has a solution if and only if $\epsilon \in \mathrm{pfx}_{\mathcal{A}}(\mathcal{G})$. In Thistle and Wonham (1994b), the set $\mathrm{pfx}_{\mathcal{A}}(\mathcal{G})$ is referred to as the "winning configurations of the supervisor" for a game theoretic interpretation.

For practical purposes, these "winning configurations of the supervisor" and, hence, the controllabilitry prefix $\mathrm{pfx}_{\mathcal{A}}(\mathcal{G})$, can be computed by a fixed-point algorithm over a product automaton induced by $\mathcal{A}$ and $\mathcal{G}$ which allows to directly extract a supervisor solving Problem 2. This algorithm is conceptually very similar to the reactive synthesis algorithm and is recalled in the following section.

Synthesis Algorithm We state a solution procedure under the assumption that $\emptyset \neq \mathcal{G} \subseteq \mathcal{A}$. By (13), this is not restrictive: if $\mathcal{G} = \emptyset$ the problem has no solution; and substitution of $\mathcal{G}$ by $\mathcal{A} \cap \mathcal{G}$ does not affect solutions. For sake of a concise exposition, we also assume that we are given a trim generalized Büchi automaton

$$M = (Q, \Sigma, q_0, \delta, \{F_{\mathcal{A}}, F_{\mathcal{G}}\}) \tag{14}$$

s.t. $\mathcal{A} = \mathrm{L}^{\omega}_{\mathrm{m}}(M_{\mathcal{A}})$, $\mathcal{G} = \mathrm{L}^{\omega}_{\mathrm{m}}(M_{\mathcal{G}})$ and $\mathrm{pfx}(\mathcal{A}) = \mathrm{L}^*(M)$ where $M_{\mathcal{A}}$ and $M_{\mathcal{G}}$ refer to the simple Büchi automaton obtained from M by using the single winning state set $F_{\mathcal{A}}$ and $F_{\mathcal{G}}$, respectively. Given the generalized Büchi automaton M, a winning configuration $s \in \mathrm{pfx}_{\mathcal{A}}(\mathcal{G})$ 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. The winning state set is computed by the four-nested fix-point

$$\nu Z\,.\,\mu Y\,.\,\nu X\,.\,\mu W\,.\,\mathsf{Pre}((W \setminus F_{\mathcal{A}}) \cup Y \cup (F_{\mathcal{G}} \cap Z), X \setminus F_{\mathcal{A}})\,, \tag{15}$$

where $\mathsf{Pre}(T,\,D)$ is the *inverse dynamics operator*, i.e.,

$$\begin{aligned}\mathsf{Pre}(T,\,D) := \{\, q \in Q \,|\, &(\exists\, \sigma \in \Sigma : \delta(q,\sigma) \in T) \,\wedge \\ &(\forall\, \sigma \in \Sigma_{\mathrm{uc}} :\ \delta(q,\sigma)! \ \Rightarrow\ \delta(q,\sigma) \in T \cup D)\,\}\,.\end{aligned} \tag{16}$$

The fixed-point in (15) is derived from the algorithm by Thistle and Wonham (1994a), which we adapted to address the special case in which both the plant $\mathcal{A}$ and the specification $\mathcal{G}$ are represented by deterministic Büchi-automata.

Given the set of winning states $\mathsf{Win}(M)$ computed via (15), we have that $\epsilon \in \mathrm{pfx}_{\mathcal{A}}(\mathcal{G})$ if and only if amounts to $q_0 \in \mathsf{Win}(M)$. Hence, the latter condition characterises the existence of solutions to Problem 2 and a supervisor can be obtained by extracting suitable control patterns from the inner loop in the fixed-point iteration.

Remark 2. For the special case in which the plant behaviour $\mathcal{A}$ is topologically closed, we can assume without loss of generality that $F_{\mathcal{A}} = Q$ in M. In this case (15) collapses to

$$\begin{aligned}\mathsf{Win}(M) &= \nu Z\,.\,\mu Y\,.\ \mathsf{Pre}(Y \cup (F_{\mathcal{G}} \cap Z)) \\ &= \nu Z\,.\,\mu Y\,.\ \mathsf{Pre}(Y) \cup (F_{\mathcal{G}} \cap \mathsf{Pre}(Z))\,,\end{aligned} \tag{17}$$

where we use the short form $\mathsf{Pre}(T) := \mathsf{Pre}(T, \emptyset)$ for the unconditional inverse dynamics operator. It should be noted that (8) and (17) are describing the same fixed-point, and this suggests a strong connection between Problem 1 and Problem 2 for the special case of a topologically closed language $\mathcal{A}$.

5. COMPARISON

This section provides a comparison between Problem 1 and Problem 2. Both problems differ in the common interpretation of how the system and the environment interact in detail. 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. In contrast, in the common interpretation in the context of supervisory control, the system passively observes past events to apply a control pattern, while the environment is responsible for the actual execution of transitions. However, both forms of interaction do not show explicitly in the formal problem statement nor in the computational solutions. 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 interpretation of system interaction are irrelevant at this stage. Using this insight, we demonstrate how one can formally transform synthesis problems 1 and 2 and their solutions into each other.

5.1 Reactive Synthesis via Supervisory Control.

In this section, we show how a solution to the reactive synthesis problem, Problem 1, can be computed using supervisory controller synthesis. This is done in three steps. Given a particular instance of Problem 1, we (i) derive a corresponding instance of Problem 2, (ii) compute a solution of the latter in terms of a non-blocking supervisor and (iii) derive a reactive module that solves the original instance of Problem 1. Step (ii) is already addressed in Sec. 4 and we are left to discuss Steps (i) and (iii).

The Corresponding Problem Given an instance of Problem 1, we are provided the non-empty finite sets U, Y, and two ω-languages $\mathcal{A} \subseteq (UY)^{\omega}$ and $\mathcal{G} \subseteq (UY)^{\omega}$ defining the assumptions on the generation of inputs and the desired guarantee on the generation of outputs, respectively. To derive a corresponding instance of Problem 2, a natural choice is to associate $\mathcal{A}$

with the plant behaviour and $\mathcal{G}$ 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, $Y = \Sigma - \Sigma_{\mathrm{uc}}$. Having set all parameters, let $f : \Sigma^* \to \Gamma$ denote a non-blocking supervisor with associated behaviour $\mathcal{L}$ that solves Problem 2.

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

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

$$h(s) := \Sigma_{\mathrm{uc}} \cup \{\sigma \in f(s) \,|\, s\sigma \in \operatorname{pfx} \mathcal{A}\} \tag{18}$$

for all $s \in \Sigma^*$. This is not expected to affect the closed-loop behaviour and, indeed, the supervisor f obtained from the algorithmic solution of Problem 2 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 $f' : \Sigma^* \to \Gamma$ where

$$f'(s) = \Sigma_{\mathrm{uc}} \dot{\cup} \{\sigma\} \text{ s.t. } h(s) \neq \Sigma_{\mathrm{uc}} \Rightarrow \sigma \in h(s) \tag{19}$$

with $\sigma \in \Sigma - \Sigma_{\mathrm{uc}}$ and for all $s \in \Sigma^*$. 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). In a third post-processing step, we intersect $\mathcal{L}'$ with $(UY)^\omega$ in order to enforce alternating inputs and outputs, i.e.,

$$\mathcal{L}'' := \mathcal{L}' \cap (UY)^\omega \,. \tag{20}$$

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 initial instance of the reactive synthesis problem.

Theorem 1. Given non-empty alphabets U, Y, $U \cap Y = \emptyset$, consider a particular instance of Problem 1, with assumption $\mathcal{A} \subseteq (UY)^\omega$ to provide the guarantee $\mathcal{G} \subseteq (UY)^\omega$. Let $f : \Sigma^* \to \Gamma$ with associated behaviour $\mathcal{L}$ be a solution to the corresponding instance of Problem 2 with $\Sigma := U \dot{\cup} Y$, $\Sigma_{\mathrm{uc}} := U$, plant behaviour $\mathcal{A}$ and specification $\mathcal{A} \cap \mathcal{G}$. Then $\mathcal{L}''$, as defined by (18)-(20), solves the given instance of Problem 1. Moreover, $\mathcal{A}$ and $\mathcal{L}''$ are non-conflicting.
If $\mathcal{L}$ is ω-regular, then f' can be chosen to be realisable by a finite automaton, and, in turn, $\mathcal{L}''$ is ω-regular. □

By the above theorem, for any instance of Problem 1 whose corresponding instance of Problem 2 has a solution in terms of a supervisor, we can use this supervisor to construct a reactive module solving the original reactive synthesis problem. For practical purposes, this amounts to solving Problem 1 via the synthesis algorithm presented in Sec. 4 (using the 4-nested fixed-point in (15)) and the additional post-processing in (18)-(20). As Thm. 1 ensures $\mathcal{A}$ and $\mathcal{L}''$ are non-conflicting, we know that the resulting reactive module will not falsify the assumptions. As mentioned in the introduction, this solution is different to solutions obtained by recently proposed algorithms synthesizing reactive modules which do not falsify the assumptions (e.g., Bloem et al. (2015); Brenguier et al. (2017)).

5.2 Supervisory Control via Reactive Synthesis

We now consider a supervisory control problem, Problem 2, and aim for a solution in terms of a reactive module. Similar to our approach in the previous Section 5.1, we proceed in three steps. Given a particular instance of Problem 2 we (i) derive a corresponding instance of Problem 1, (ii) compute a solution of the latter in terms of a reactive module and (iii) derive a non-blocking supervisor that solves the original instance of Problem 2. Unfortunately, step (iii) cannot be performed in general, as a non-blocking supervisor by definition requires that $\mathcal{L}$ and $\mathcal{A}$ are non-conflicting and this can not be addressed by an ω-regular specification in the RS problem under consideration. Hence, we propose to impose additional assumptions on the problem parameters that ensure that the instance of Problem 1 obtained in step (i) is such that the associated behaviour $\mathcal{L}$ of the resulting reactive module does not conflict with $\mathcal{A}$. Interestingly, there exist two conditions which where developed independently in both communities and which effectively ensure a non-conflicting closed loop. Both conditions address the special class of plant behaviours $\mathcal{A} \subseteq (\Sigma_{\mathrm{uc}}(\Sigma - \Sigma_{\mathrm{uc}}))^\omega$ with alternating controllable and uncontrollable events, were all controllable events $\Sigma - \Sigma_{\mathrm{uc}}$ are a locally free input to $\mathcal{A}$. We restrict the discussion in this section to this system class, referred to as *input/output plant behaviours with locally free input.*

The Corresponding Problem Given an instance of Problem 2 we are provided with $\Sigma_{\mathrm{uc}} \subsetneq \Sigma$, an input/output plant behaviour $\mathcal{A}$ with locally free input, and a specification $\mathcal{G} \subseteq \Sigma^\omega$. By the alternation of controllable and uncontrollable events, we can choose the correspondence $U := \Sigma_{\mathrm{uc}}$ and $Y := \Sigma - \Sigma_{\mathrm{uc}}$ to obtain $\mathcal{A}$, $G \subseteq (UY)^\omega$. Excluding the trivial case $\Sigma_{\mathrm{uc}} = \Sigma$, this constitutes qualifying parameters for the reactive synthesis problem, Problem 1. For the following, we consider a reactive module with behaviour $\mathcal{L}$ that solves Problem 1 and seek to construct a solution to the initial instance of the supervisory control problem, Problem 2.

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

$$\mathcal{L}' := \mathcal{L} \cup ((\operatorname{pfx} \mathcal{L})(\Sigma_{\mathrm{uc}}^\omega)) \,. \tag{21}$$

The above construct $\mathcal{L}'$ indeed satisfies (SC1) and (SC2) and, moreover, retains the closed-loop behaviour, i.e., $\mathcal{L}' \cap \mathcal{A} = \mathcal{L} \cap \mathcal{A}$ and $(\operatorname{pfx} \mathcal{L}') \cap (\operatorname{pfx} \mathcal{A}) = (\operatorname{pfx} \mathcal{L}) \cap (\operatorname{pfx} \mathcal{A})$.

For $\mathcal{L}'$ to solve Problem 2, 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 $\emptyset \neq \mathcal{A} \cap \mathcal{L}' \subseteq \mathcal{G}$.

Non-Falsifiable Assumptions in Reactive Synthesis As the reactive module with behaviour $\mathcal{L}$ solves Problem 1 it fulfils (3), and hence, the closed-loop configuration can continue for infinitely many computation cycles to generate an ω-word $\alpha \in$

$(\mathrm{clo}\,\mathcal{A}) \cap (\mathrm{clo}\,\mathcal{L})$. Since $\mathcal{L}$ is closed, we also have $\alpha \in \mathcal{L}$. However, one may fail on $\alpha \in \mathcal{A}$, and, by the specification $\mathcal{L} \subseteq \mathcal{A} \to \mathcal{G}$, risk that $\alpha \notin \mathcal{G}$. This undesirable situation is referred to as *falsifying the assumption*.

This issue can be avoided by *non-falsifiable assumptions*[5]. Given the two player game interpretation used in the algorithmic synthesis of reactive modules, an assumption cannot be falsified by the module, 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 attained 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 As Problem 2 asks 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 require that $s \in \mathrm{pfx}(\mathcal{A} \cap \mathcal{L}')$ and the existence of $\beta \in \Sigma^\omega$ 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. See Moor et al. (2011) for an example that illustrates this subtle issue.

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

$$\mathrm{pfx}_{\mathrm{clo}\,\mathcal{A}}(\mathcal{A}) = \mathrm{pfx}\,\mathcal{A} \tag{22}$$

where $Y = \Sigma - \Sigma_{uc}$ take the role of the uncontrollable events in the definition of the controllability prefix, Def. 2. The latter property is referred to as *strong non-anticipation*. Referring to the game theoretic interpretation of supervisory control, (22) requires that the *local plant* $\mathrm{pfx}\,\mathcal{A}$ is always in a winning configuration regarding the satisfaction of its own acceptance condition, i.e., at any time the plant can decide to internally use 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.

Result When comparing the two conditions, namely strong non-anticipation and non-falsifiable assumptions, one can show that the former is stronger then the latter. An intuition for this result can be obtained by comparing the game theoretic interpretations of both conditions. A non-falsifiable assumption requires the “environment to play clever” from the very beginning, whereas (22) allows the plant to start doing so eventually. We provide a more technical comparison in Schmuck et al. (2017). Our second main result is stated as follows.

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

Theorem 2. Given an alphabet Σ with the non-empty set of uncontrollable events $\Sigma_{\mathrm{uc}} \subsetneq \Sigma$, consider a particular instance of Problem 2 with input/output plant behaviour $\mathcal{A}$ with locally free input, and a specification $\mathcal{G} \subseteq \mathcal{A}$. Let r with associated behaviour $\mathcal{L}$ be a solution to the corresponding instance of Problem 1 with $U = \Sigma_{\mathrm{uc}}$ and $Y = \Sigma - \Sigma_{\mathrm{uc}}$, assumptions $\mathcal{A}$ and guarantee $\mathcal{G}$. If $\mathcal{A}$ satisfies (22), the assumptions are non-falsifiable and $\mathcal{L}'$ defined by (21) solves the given instance of Problem 2.
If $\mathcal{L}$ is ω-regular, then so is $\mathcal{L}'$. □

By the above theorem we have the following result. Given an instance of Problem 2 with a plant behaviour $\mathcal{A}$ which (i) is strongly non-anticipating, (ii) has alternating controllable and uncontrollable events, and (iii) where the controllable events are a locally free input to $\mathcal{A}$, any solution to the corresponding instance of Problem 1 can be used to construct a non-blocking supervisor solving the original supervisory controller synthesis problem. For practical purposes, this amounts to solving Problem 2 via the reactive synthesis algorithm presented in Sec. 3 (using the 3-nested fixed-point in (8)) and the additional post-processing in (21).

Remark 3. It should be noted that topologically closed plant behaviours are always strongly non-anticipating. Hence, combining the results form Thm. 1 and Thm. 2 with the observations from Rem. 1 and Rem. 2 we see that for topologically closed plants with input/output behaviour both problems are essentially equivalent and can be solved by the matching two-nested fixed-points in (8) and (17).

Remark 4. A more general discussion for behaviours that do not necessarily have an input/output structure is given in Schmuck et al. (2017) and leads to a technically more involved transformation when solving an SCT synthesis problem via RS. This transformation amounts to using control patterns as outputs and generated events as inputs to the reactive module and requires a more subtle translation between supervisors and reactive modules. In this situation, we can show that topological closedness of $\mathcal{A}$ is a prerequisite to ensure that the resulting closed-loop is non-conflicting. This implies that the result stated in Rem. 3 also holds for behaviours that do not necessarily have an input/output structure.

6. CONCLUSION

This paper provides a formal comparison of reactive synthesis under environment assumptions, Problem 1, and supervisory control of input/output behaviours, Problem 2.

As our first main result, we have shown in Sec. 5.1 (Thm. 1) that one can solve the considered RS problem, Problem 1, by using supervisory controller synthesis to obtain a reactive module which will not actively falsify the assumptions on the environment. This provides a new perspective on algorithms ensuring soltutions with this property, which is an active field of research in the RS comunity.

As our second main result, we have shown in Sec. 5.2 (Thm. 2) that the synthesis problem from SCT, Problem 2, with strongly non-anticipating and input/output plant behaviour can be solved using RS. Both properties are well motivated for hybrid systems or abstractions thereof, see Moor et al. (2011), and strong non-anticipation is conceptionally and technically closely related to non-falsifiable assumptions. Using RS for supervisory controller synthesis for this class of plants demonstrates that a three nested fixed-point computation suffices. We may therefore expect computational benefits as a trade-off when imposing additional conditions on the synthesis problem from SCT.

In conjunction, Thms. 1 and 2 establish the equivalence of the two problems regarding solvability for the subclass of strongly non-anticipating plants with locally free input.

ACKNOWLEDGEMENTS

We thank Nir Piterman for insightful discussions about the supervisory controller synthesis algorithm.

REFERENCES

Bloem, R., Ehlers, R., Jacobs, S., and Könighofer, R. (2014). How to handle assumptions in synthesis. In *SYNT*, 34–50.

Bloem, R., Ehlers, R., and Könighofer, R. (2015). Cooperative reactive synthesis. In *ATVA 2015*, 394–410.

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

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

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

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

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

Cai, K., Zhang, R., and Wonham, W.M. (2015). Relative observability of discrete-event systems and its supremal sublanguages. *IEEE Trans. Autom. Control*, 60(3), 659–670.

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

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

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

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

Cury, J. and Krogh, B. (1999). Robustness of supervisors for discrete-event systems. *TAC*, 44, 376–379.

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

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

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

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

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

Hopcroft, J.E. and Ullman, J.D. (1979). *Introduction to Automata Theory, Languages and Computation*.

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

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

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

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

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

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., and Lafortune, S. (2011). Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, 47(4), 639–649.

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

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

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

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

Ramadge, P.J. and Wonham, W.M. (1989). Modular control of discrete event systems. *Maths. of Cont., Sign. & Sys.*, 13–30.

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

Schmidt, K., Moor, T., and Perk, S. (2008). Nonblocking hierarchical control of decentralized discrete event systems. *IEEE Trans. Autom. Control*, 53(10), 2252–2265.

Schmuck, A.K., Moor, T., and Majumdar, R. (2017). On the relation between reactive synthesis and supervisory control of non-terminating processes. Technical report, MPI-SWS.

Thistle, J.G. and Wonham, W.M. (1994a). Control of infinite behavior of finite automata. *SIAM Cont. & Opt.*, 1075–1097.

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

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

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

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

Wen, Q., Kumar, R., Huang, J., and Liu, H. (2008). A framework for fault-tolerant control for discrete event systems. *IEEE Trans. Autom. Control*, 53, 1839–1849.

Willems, J.C. (1991). Paradigms and puzzles in the theory of dynamic systems. *Trans. on Autom. Control*, 36, 258–294.

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

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

Zhong, H. and Wonham, W.M. (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. *Theoretical Computer Science*, 200(1-2), 135–183.