Synthesis of Distributed and Decentralised Supervisory Control via Contract Negotiation

Ana Maria Mainhardt and Anne-Kathrin Schmuck

***Abstract*— We propose a method for the synthesis of *local decentralised* supervisors for *distributed* discrete event systems where subsystems may have different controllability settings for their shared events, and where the *privacy* of the behaviour in terms of the local nonshared events plays a vital role. For simplicity, only two subsystems are considered. Our approach is based on the negotiation of contracts represented solely over shared events and describing an agreement between the subsystems, such that they cooperate in disabling events in order to guarantee their local specifications are satisfied. We establish further conditions to ensure correct cooperation and global nonblockingness, which are locally checked and, if necessary, enforced. This way, privacy is respected during both the synthesis and the execution of the supervisors, as the dynamics of a subsystem are only partially observed by the other through the events they actually share, without the communication of exclusively local events or the use of a coordinator for their supervisors. To protect privacy, maximal permissiveness may be sacrificed; nevertheless, we identify cases where this trait is preserved.**

***Index Terms*— Automata, Discrete Event Systems, Privacy, Supervisory Control**

I. Introduction

Assume-Guarantee (A/G) contracts are a well established paradigm for dealing with large-scale distributed systems and have a longstanding history and well established theoretical foundations – see e.g. [1], [2]. Its main idea is to decouple dependent distributed processes, or subsystems, by making use of a set of *compatible local contracts* consisting of an assumption and a guarantee. If the rest of the system fulfils a process' assumption, this process must ensure its local guarantee holds, which in turn implies that the corresponding assumptions induced on the others also hold. The implications of this methodology are very appealing from a practical perspective, as they allow for (i) efficient design, i.e., local controllers can be synthesised in a decentralised and concurrent fashion; (ii) information privacy, meaning that, apart from what is specified by the contracts, no detailed information about a local behaviour is shared with the rest of the plant; and (iii) decoupled maintenance, as contract-compatible adaptations in a component do not affect others.

Motivated by these desirable features, particularly privacy, this paper presents an assume-guarantee synthesis framework for the decentralised supervisory control of distributed discrete event system. This requires the design of local supervisors respecting compatible contracts, which model the common behaviour of the supervised processes solely in terms of the events they share, i.e, synchronise over. Each supervisor has only partial observation of the events from the rest of the system. Therefore, our framework must also identify and be able to enforce local conditions that are sufficient to guarantee cooperative control and nonblocking global behaviour of the resulting closed-loop system. Due to page constraints, this paper focuses on the specific but relevant case of two distributed processes, and leaves its intricate extension to multiple processes for future work.

The method we propose here is inspired by the recent work of [3] in the context of reactive synthesis, and is based on the *negotiation* of contracts. The idea is the following. An initial local supervisor is designed for each process by assuming some cooperation from the rest of the plant, and then translated into contracts to be exchanged between the subsystems. Iteratively, our procedure refines the local supervisors and generates new contracts to be negotiated. Termination occurs when compatible contracts and supervisors respecting these contracts are achieved, and the aforementioned conditions are locally guaranteed. Our framework results in local supervisors correctly enforcing a global behaviour without a coordinator, and without communicating exclusively local events during both the synthesis and the execution of these supervisors.

A. Motivating Example

To illustrate our framework, consider the following example, as depicted in Fig. 1, where we have two distinct manufacturing lines: (0) on top and (1) on bottom. Assume that each line is from a different vendor, and has its own specifications; the details of the dynamics of each line are irrelevant here for our motivation. These lines are connected through buffers that operate in a first in, first out manner, and that are used to send pieces being processed in one line to the other one. The events of putting and retrieving pieces of certain type from each of these buffers are therefore shared events, and directly affect the behaviour of both lines. However shared, each of them may be controllable by only one of the lines. For example,

Manuscript received (date); revised (dates); accepted (date). Date of publication (date); date of current version (date). Both authors are funded through the DFG Emmy Noether grant SCHM 3541/1-1. A. Schmuck is also partially funded through the DFG collaborative research center 389792660 TRR 248 – CPEC.

A. Mainhardt and A. Schmuck are with the Max-Planck Institute for Software Systems, Kaiserslautern, Germany (e-mail: {amainhardt,akschmuck}@mpi-sws.org).

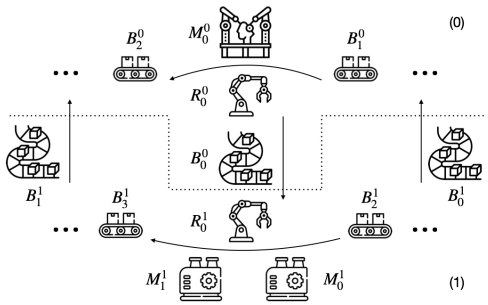

Fig. 1. Illustrative example of two manufacturing lines, $(\mathbf{0})$ on top, $(\mathbf{1})$ on bottom. Symbols $\boldsymbol{B_k^i}$, $\boldsymbol{R_k^i}$ and $\boldsymbol{M_k^i}$ indicate, respectively, the $\boldsymbol{k}$-th buffer, robot and machine of line $(\boldsymbol{i})$. Arrows represent the flow of pieces in each line and between them, and ellipsis represent omitted parts of the plant. The robots take the pieces from buffers to the machines to be processed, and then to the next buffers. Details of the dynamics of the lines are not important, just that they synchronise over the shared events related to the buffers that send pieces from one line to the other.

line (0) can control the event of putting a workpiece in the central buffer, while it cannot control the event of taking it from the other side; vice-versa for line (1).

It is realistic to consider the scenario where each vendor wishes to hide from the other the details of their own line's dynamics, that is, everything in their behaviour concerning their local nonshared events, that is, their private events. The idea of our approach is, then, to allow *cooperation* in disabling shared events that are controllable by one, but not by the other, and to design *compatible contracts* as well as local supervisors that respect them. These contracts represent an agreement between these vendors, but purely in terms of their shared events – for example, specifying the capacity of such buffers, or the order in which different types of workpieces are sent through them – in a way that both lines satisfy their local specifications and conflict is prevented.

B. Related Work and Contribution

Due to the decentralised nature of the resulting local supervisors, our framework has to cope with similar challenges as other decentralised, distributed, or modular supervisor synthesis techniques. One of the main challenges is to ensure *nonconflict*, an intrinsically global property stating that the entire plant in closed loop with its decentralised controllers can always reach desired, or *marked*, states. Some of these approaches require this property to be verified over the composition of the controlled subsystems – e.g., [4], [5] – bringing a computational cost that, in the worst case, is of the same order as that of the monolithic approach [6]. There are works, such as that of [7], in which such verification over the entire model is not required; this is done by establishing sufficient conditions for nonconflict over the local components, such that the verification process also becomes decentralised. Either way, if conflict is detected, it needs to be resolved somehow. One way is by designing a *coordinator*, i.e., an additional supervisor that coordinates the control actions of the local ones – see [8]. Alternatively, if it is feasible to *communicate* the occurrence of events which are a priori nonshared from one supervisor to another – by the addition of sensors, for example – then it is possible to determine a set of external events that each local supervisor needs to observe to guarantee nonconflict – e.g. [9]–[11], to cite a few.

Our method, on the other hand, guarantees *nonconflict by construction* without a coordinator or communication of exclusively local events, being a suitable solution when this sort of communication is not desirable for security reasons, or is not physically or financially viable. To ensure nonconflict, we introduce here a new, targeted property called *relative unambiguity*, inspired by existing conditions for decomposability, such as relative observability from [12], and locally nonblocking condition from [7]. In addition, we adopt the cooperative controllability setting from [9], and the *halfway synthesis* concept from [13] to obtain *maximally permissive* local supervisors whenever the aforementioned property is immediately satisfied, i.e., when it does not need to be enforced.

Our framework is closely related to other iterative A/G-based synthesis approaches from the formal methods community, e.g. [3], [14], [15]. The particularity of our work in this context is the use of synchronised deterministic finite automata (DFA) over *finite* words as system models, compared to input/output ω-automata over *infinite* words. This brings interesting consequences. Firstly, to the best of our knowledge, there does not yet exist a sound assume-guarantee framework allowing ω-automata with arbitrary markings as contracts: existing work only uses fully marked (safety) automata for this purpose. Our method circumvents this problem, as the synchronisation of DFA ensures that liveness requirements, encoded by marked states in the involved automata, must be enforced by all subsystems at the same time. Secondly, synchronisation of automata leads to blocking situations in a distributed setting – which requires special attention – while interacting input/output ω-automata never block.

Note that this paper subsumes its conference predecessor [16], extending the contributions in the following ways. The results are presented and discussed in a more didactical and thorough manner, and the proofs, which were omitted in the aforementioned paper, are presented in full here. Moreover, in Sec. V-C, we redefine the local property for nonconflict, *relative unambiguity*, by relaxing it. This is possible by allowing the supervisors to cooperate via signalling bits related to the events they share, but that are controllable for one and not for the other, if that is the case in the system; in Sec. IV-E we discuss why these signalling bits differ from the communication of private events. Finally, while in [16] we assume that the events shared among the subsystems composing a plant are controllable by at least one of them, here we treat the more general case by assuming some of these shared events are not controllable by any part of the plant. This extension not only raises interesting discussions, but it requires special attention and substantial changes in our framework, leading to a new procedure for our *Contract-Based Supervisor Synthesis* approach.

II. Preliminaries

In this section, we briefly introduce the notation and definitions relevant for this document. For a more didactic coverage on supervisory control, see the textbooks [17] and [8].

A. Languages and Automata Basics

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

Sets. For any two sets A and B, denote by $A \times B$ and $A \backslash B$, respectively, their Cartesian product and set difference.

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

Projections. Given two alphabets Σ and $\Sigma_i \subseteq \Sigma$, the *natural projection* $P_i : \Sigma^* \to \Sigma_i^*$ is defined recursively by: (i) $P_i(\epsilon) = \epsilon$, and (ii) for all $\sigma \in \Sigma$ and $s \in \Sigma^*$, if $\sigma \in \Sigma_i$, then $P_i(s\sigma) = P_i(s)\sigma$, otherwise $P_i(s\sigma) = P_i(s)$. We define the *inverse projection* $P_i^{-1} : \Sigma_i^* \to 2^{\Sigma^*}$ by $P_i^{-1}(s_i) = \{s \in \Sigma^* \mid P_i(s) = s_i\}$ for all $s_i \in \Sigma_i^*$. These functions can be extended from strings to languages, with $P_i : 2^{\Sigma^*} \to 2^{\Sigma_i^*}$ defined such that, for any $L \subseteq \Sigma^*$, $P_i(L) = \{s_i \in \Sigma_i^* \mid \exists s \in L : P_i(s) = s_i\}$. In turn, $P_i^{-1} : 2^{\Sigma_i^*} \to 2^{\Sigma^*}$ is given by $P_i^{-1}(L_i) = \{s \in \Sigma^* \mid P_i(s) \in L_i\}$ for any $L_i \subseteq \Sigma_i^*$.

Automata. A *deterministic finite automaton* (*automaton* for short) is a tuple $\mathbf{\Lambda} = (Q, \Sigma, \delta, q_0, Q_\mathrm{m})$, with finite *state set* Q, *initial state* $q_0 \in Q$, *marked states* $Q_\mathrm{m} \subseteq Q$, and the *deterministic transition function* $\delta : Q \times \Sigma \to Q$, which can also be viewed as a relation $\delta \subseteq Q \times \Sigma \times Q$. This function is *partial* if $\delta(q, \sigma)$ is not defined for all $q \in Q$ and $\sigma \in \Sigma$; otherwise, we say it is *total*. We identify δ with its extension to the domain $Q \times \Sigma^*$, or as a relation $\delta \subseteq Q \times \Sigma^* \times Q$. Let $\delta(q, s)!$ indicate that δ is defined for $q \in Q$ and $s \in \Sigma^*$; for all $q \in Q$, we have $\delta(q, \epsilon) = q$; for $s \in \Sigma^*$ and $\sigma \in \Sigma$, we have $\delta(q, s\sigma)!$, with $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma))$, if and only if $\delta(q, s)!$ and $\delta(\delta(q, s), \sigma))!$. We say any (well-defined) automaton $\mathbf{\Lambda}$ is *nonempty*, unless it is what is termed *the empty automaton* ([17]), where $Q = \emptyset$ and q_0 is not defined.

Reachability and Nonblockingness. A state $q \in Q$ is *reachable* if there exists $s \in \Sigma^*$ such that $q = \delta(q_0, s)$, and it is *coreachable* if there exists $s \in \Sigma^*$ such that $\delta(q, s) \in Q_\mathrm{m}$. Non coreachable states are also referred to as *blocking states*. If all reachable states in $\mathbf{\Lambda}$ are coreachable, then $\mathbf{\Lambda}$ is *nonblocking*. Moreover, $\mathbf{\Lambda}$ is called *reachable* (respectively *coreachable*) if all states are reachable (resp. coreachable), and $\mathbf{\Lambda}$ is called *trim* if it is reachable and coreachable. The *reachable* (resp. *coreachable*) subautomaton $\mathrm{Re}(\mathbf{\Lambda})$ (resp. $\mathrm{CoRe}(\mathbf{\Lambda})$) is obtained by eliminating all nonreachable (resp. blocking) states from $\mathbf{\Lambda}$ and, accordingly, adapting the transition function δ in the obvious way. The *trim subautomaton* $\mathrm{Trim}(\mathbf{\Lambda})$ is given by $\mathrm{Re}(\mathrm{CoRe}(\mathbf{\Lambda})) = \mathrm{CoRe}(\mathrm{Re}(\mathbf{\Lambda}))$.

Semantics. For $\mathbf{\Lambda} = (Q, \Sigma, \delta, q_0, Q_\mathrm{m})$, we associate the *generated language* $\mathcal{L}(\mathbf{\Lambda}) := \{ s \in \Sigma^* \mid \delta(q_0, s)! \}$ and the *marked language* $\mathcal{L}_m(\mathbf{\Lambda}) := \{ s \in \Sigma^* \mid \delta(q_0, s) \in Q_\mathrm{m} \}$. These two languages are the *semantics* of $\mathbf{\Lambda}$, and $\mathbf{\Lambda}$ *recognizes*, or *accepts*, the language $\mathcal{L}_m(\mathbf{\Lambda})$. Moreover, we say any two automata $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$ are *equivalent* if their semantics coincide, i.e., we write $\mathbf{\Lambda}_1 \equiv \mathbf{\Lambda}_2$ if and only if $\mathcal{L}(\mathbf{\Lambda}_1) = \mathcal{L}(\mathbf{\Lambda}_2)$ and $\mathcal{L}_m(\mathbf{\Lambda}_1) = \mathcal{L}_m(\mathbf{\Lambda}_2)$; if they are not equivalent, we write $\mathbf{\Lambda}_1 \not\equiv \mathbf{\Lambda}_2$. Note that $\mathbf{\Lambda}$ is nonblocking if and only if $\mathcal{L}(\mathbf{\Lambda}) = \overline{\mathcal{L}_m(\mathbf{\Lambda})}$. To simplify notation, we use boldface characters, e.g. $\mathbf{\Lambda}$, to denote an automaton, and the corresponding normal character, e.g. Λ, for its marked language.

Product and Composition. The *synchronous product* of languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, where Σ_i, for $i \in \{1, 2\}$, are arbitrary alphabets, is defined as $L_1 \parallel L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$ over $\Sigma_1 \cup \Sigma_2$. Given two automata $\mathbf{\Lambda}_i$ over Σ_i, their *synchronous composition* $\mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2$ is defined such that $\mathcal{L}(\mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2) = \mathcal{L}(\mathbf{\Lambda}_1) \parallel \mathcal{L}(\mathbf{\Lambda}_2)$ and $\mathcal{L}_m(\mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2) = \Lambda_1 \parallel \Lambda_2$. That is, only shared events $\sigma \in \Sigma^\mathrm{s} = \Sigma_1 \cap \Sigma_2$ require synchronisation of the corresponding transitions.

B. Supervisory Control

Plant Model. A plant is a system to be supervised, and it is modelled as an automaton $\mathbf{M}$ over Σ, which can be partitioned as $\Sigma = \Sigma_\mathrm{c} \dot\cup \Sigma_\mathrm{uc}$, where Σ_c is the set of *controllable* events – those that can be prevented from happening, or disabled – and Σ_uc is the set of *uncontrollable* ones – that cannot be disabled.

Specification. A set of *specifications* for a plant $\mathbf{M}$ over the alphabet Σ is modelled by trim automata $\mathbf{E}_k$ over $\Sigma_{\mathbf{E},k} \subseteq \Sigma$, where k is an index to represent each element in the set. The purpose of a specification $\mathbf{E}_k$ is to model merely which sequences of events in $\Sigma_{\mathbf{E},k}$ are allowed to occur in the closed-loop behaviour, without taking into account the sequences that are indeed possible in the free (open-loop) behaviour of $\mathbf{M}$. A plant can be composed with a set of specifications, and the resulting automaton is then called its *desired behaviour*. Here, we consider that each specification does not alter the marking of the plant, that is, all the states in $\mathbf{E}_k$ are marked, for all k in the set. It is useful to use a single automaton to represent both the plant and its set of specifications, as follows. Denote by $\mathbf{E}$ over Σ' the composition of all the specifications $\mathbf{E}_k$ in the set, with $\Sigma' \subseteq \Sigma$ being the union of all the alphabets $\Sigma_{\mathbf{E},k}$. Next, for $\mathbf{E} = (Q, \Sigma', \delta, q_0, Q)$, we define $\mathbf{E}' = (Q \cup \{\bot\}, \Sigma', \delta^\bot, q_0, Q)$ such that $\delta^\bot = \delta \cup \big\{(x, \sigma, \bot) \in Q \times \Sigma' \times \{\bot\} \mid \delta(x, \sigma) \text{ is not defined}\big\}$. The automaton $\mathbf{K} = \mathbf{M} \parallel \mathbf{E}'$ is a *plantified specification* for $\mathbf{M}$. We have that $\mathbf{K}$ is a blocking automaton whose semantics are contained in the semantics of the plant, and whose marked language is M-closed. Please note that *specification* and *plantified specification* are not interchangeable terms.

Controllability. A language L over Σ is *controllable* with respect to $\mathcal{L}(\mathbf{M})$ and Σ_uc if, for all $\sigma \in \Sigma_\mathrm{uc}$, $s \in \overline{L} \wedge s\sigma \in \mathcal{L}(\mathbf{M}) \Rightarrow s\sigma \in \overline{L}$. Define the set $\mathcal{C}(L) := \{L' \subseteq L \mid L' \text{ is controllable w.r.t. } \mathcal{L}(\mathbf{M}) \text{ and } \Sigma_\mathrm{uc}\}$, whether L is controllable or not. This set is nonempty, since the empty language is trivially controllable. As controllability is closed under union of languages, it can be shown that the supremum element of $\mathcal{C}(L)$, denoted $\sup\mathcal{C}(L)$, is given by $\bigcup_{L' \in \mathcal{C}(L)} L'$ and is controllable, i.e., belongs to $\mathcal{C}(L)$.

Supervisor. A *supervisor* for a plant $\mathbf{M}$ with alphabet $\Sigma := \Sigma_\mathrm{c} \dot\cup \Sigma_\mathrm{uc}$ is a mapping $f : \mathcal{L}(\mathbf{M}) \to \Gamma$, where $\Gamma := \{\gamma \subseteq$

$\Sigma \,|\, \Sigma_{\rm uc} \subseteq \gamma\} \subseteq 2^{\Sigma}$ is the set of *control patterns*. Let us denote by $f/\mathbf{M}$ the plant $\mathbf{M}$ under supervision of f. The generated language of $f/\mathbf{M}$ is defined recursively such that $\epsilon \in \mathcal{L}(f/\mathbf{M})$ and, for all $s \in \Sigma^*$ and $\sigma \in \Sigma$, $s\sigma \in \mathcal{L}(f/\mathbf{M})$ iff (i) $s \in \mathcal{L}(f/\mathbf{M})$, (ii) $s\sigma \in \mathcal{L}(\mathbf{M})$, and (iii) $\sigma \in f(s)$. This induces the marked language $\mathcal{L}_m(f/\mathbf{M}) := \mathcal{L}(f/\mathbf{M}) \cap M$, which is controllable by definition. Note that $\mathcal{L}(f/\mathbf{M})$ is closed and $\mathcal{L}(f/\mathbf{M}) \subseteq \mathcal{L}(\mathbf{M})$. We call f nonblocking if $\mathcal{L}(f/\mathbf{M}) = \overline{\mathcal{L}_m(f/\mathbf{M})}$. In this case, the closed loop can be represented by a trim automaton $\mathbf{S}$ that accepts $\mathcal{L}_m(f/\mathbf{M})$; we then say that $\mathbf{S}$ *realises* $f/\mathbf{M}$ and denote this by $\mathbf{S} \sim f$.

Supervisor Synthesis Problem. Given a plant $\mathbf{M}$ and a plantified specification $\mathbf{K}$ over $\Sigma := \Sigma_{\rm c} \dot\cup \Sigma_{\rm uc}$, the control problem is to design the *maximally permissive* supervisor f that respects the specification – i.e., $\mathcal{L}_m(f/\mathbf{M}) = \sup\mathcal{C}(K)$, with controllability taken with respect to $\mathcal{L}(\mathbf{M})$ and $\Sigma_{\rm uc}$ – and that imposes a nonblocking closed-loop behaviour – i.e., $\mathcal{L}(f/\mathbf{M}) = \overline{\mathcal{L}_m(f/\mathbf{M})}$.

Supervisor Computation. There exist many different algorithms to solve the stated synthesis problem. In particular, it is possible to compute a trim automaton $\mathbf{S} \sim f$ by manipulating a single automaton – namely, the plantified specification $\mathbf{K}$ – instead of multiple automata – i.e., the plant $\mathbf{M}$ separately from the (nonplantified) specifications. See [17] for details. Hence, let us define the function SYNTH.

Definition 1: For an automaton $\mathbf{S}^0$ over an alphabet $\Sigma := \Sigma_{\rm c} \dot\cup \Sigma_{\rm uc}$, we define the function SYNTH such that $\text{SYNTH}(\mathbf{S}^0) = \mathbf{S}$, where $\mathbf{S}$ is a trim automaton that accepts the language $\sup\mathcal{C}(S^0)$ with respect to $\mathcal{L}(\mathbf{S}^0)$ and $\Sigma_{\rm uc}$. □

Remark 1: If $\mathbf{K}$ is a plantified specification for a plant $\mathbf{M}$ over $\Sigma := \Sigma_{\rm c} \dot\cup \Sigma_{\rm uc}$, then $\sup\mathcal{C}(K)$ with respect to $\mathcal{L}(\mathbf{K})$ and $\Sigma_{\rm uc}$ is equal to $\sup\mathcal{C}(K)$ with respect to $\mathcal{L}(\mathbf{M})$ and $\Sigma_{\rm uc}$. □

III. CONTRACT-BASED SYNTHESIS PROBLEM

This section introduces the distributed and decentralised supervisor synthesis problem tackled in this paper, and formalises the concept of assume-guarantee for this context.

A. Decentralised Supervisor Synthesis Problem Setting

We consider a plant composed of two processes, or subsystems, both modelled as automata $\mathbf{M}_i$ over alphabets Σ_i, $i \in \{1, 2\}$. Along this paper, the terms *global* and *local* are used to refer to the plant and its subsystems, respectively. Each local alphabet is partitioned into $\Sigma_i = \Sigma_i^{\rm p} \dot\cup \Sigma^{\rm s}$, where $\Sigma^{\rm s}$ is the set of *shared* events – i.e., $\Sigma^{\rm s} = \Sigma_1 \cap \Sigma_2$ – and $\Sigma_i^{\rm p}$ is the set of *private* events of $\mathbf{M}_i$, which are its local and exclusive events, that is, $\Sigma_i^{\rm p} = \Sigma_i \setminus \Sigma^{\rm s}$. We assume that $\Sigma_i^{\rm p}$ and $\Sigma^{\rm s}$ may contain controllable and uncontrollable events, meaning $\Sigma_i^{\rm p} = \Sigma_{i,\rm c}^{\rm p} \dot\cup \Sigma_{i,\rm uc}^{\rm p}$ and $\Sigma^{\rm s} = \Sigma_{\rm c}^{\rm s} \dot\cup \Sigma_{\rm uc}^{\rm s}$. We define the natural projections $P_{i\rm s} : \Sigma_i^* \to \Sigma^{\rm s*}$ and $P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*$. To simplify notation throughout the document, we use the convention that the indices i, j are understood from the context such that $i \neq j$ for all $i, j \in \{1, 2\}$.

We do *not* require both processes to agree on the controllability status of shared events. From a global perspective of the plant, we have that shared events controllable by at least one subsystem are globally controllable, thus belonging to $\Sigma_{\rm c}^{\rm s}$. That is, $\Sigma_{\rm c}^{\rm s} = \Sigma_{1,\rm c}^{\rm s} \cup \Sigma_{2,\rm c}^{\rm s}$, where $\Sigma_{i,\rm c}^{\rm s} := \Sigma_{i,\rm c} \backslash \Sigma_i^{\rm p}$, and $\Sigma_{i,\rm c}$ is the set of all events controllable by $\mathbf{M}_i$. The subset of $\Sigma_{\rm c}^{\rm s}$ that is locally uncontrollable to i, thus controllable to j, we denote by $\Sigma_{i,\rm cp}^{\rm s} = \Sigma_{\rm c}^{\rm s} \backslash \Sigma_{i,\rm c}^{\rm s}$, where cp stands for *cooperation*. For the cases where $\Sigma_{i,\rm cp}^{\rm s} \neq \emptyset$ for some i – when the supervisors do not agree on the controllability status of all the events they share – and in order to allow cooperation and joint control ([18]), we define the set $\Sigma_i^{\rm d} = \{{\rm d}^\sigma \,|\, \sigma \in \Sigma_{i,\rm cp}^{\rm s}\}$, where ${\rm d}^\sigma$ represents the request from supervisor i to j to disable a shared event σ in $\Sigma_{i,\rm cp}^{\rm s}$. The shared events no subsystem can control are globally uncontrollable, belonging to $\Sigma_{\rm uc}^{\rm s}$; in [16], we considered this set as empty, i.e., we assumed that all shared events are controllable by at least one subsystem.

Finally, we consider that each process is equipped with a plantified specification $\mathbf{K}_i$ over Σ_i.

B. Contract-based Supervisor Synthesis.

Problem. For the problem setting given above, we wish to find local supervisors $f_i : \mathcal{L}(\mathbf{M}_i) \times 2^{\Sigma_j^{\rm d}} \to \Gamma_i$ such that the global closed-loop behaviour satisfies the local specifications and is nonblocking, that is, $\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2) \subseteq K_1 \parallel K_2$ and $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2)}$, where $\mathcal{L}(f_i/\mathbf{M}_i)$ and $\mathcal{L}_m(f_i/\mathbf{M}_i)$ are defined from f_i as in Sec. II-B.

Assume-Guarantee Contracts. The underlying idea of this paper is to use *contracts* to represent a cooperative *agreement* between decentralised components. Since each local process can observe the other's dynamics solely by the occurrence of events they share, contracts are defined by automata over such events and they represent both what one local supervisor expects from and what it promises to the other controlled process in terms of disabling those events. To formalise this, let us initially model a contract for the subsystem $\mathbf{M}_i$ as a tuple of automata $(\mathbf{A}_i, \mathbf{G}_i)$ over the shared alphabet $\Sigma^{\rm s}$.

The automaton $\mathbf{G}_i$ represents the *guarantee* the subsystem i needs to provide for the rest of the plant. It is an additional local *specification*, in the sense defined in Sec. II-B: it solely informs which sequences of events – in this case, shared ones – are allowed to occur in the local closed-loop behaviour, disregarding which ones are actually generated by $\mathbf{M}_i$.

The automaton $\mathbf{A}_i$ is the *assumption*. It models the closed-loop behaviour of the rest of the plant as perceived by subsystem i. Thus, the local supervisor f_i does not enforce over its subsystem the restrictions that are needed to satisfy its local specifications but that are assumed to be *already imposed* by the rest of the plant – since all controlled subsystems synchronise over shared events in the global dynamics.

Due to the synchronisation over $\Sigma^{\rm s}$, and in order to compute each supervisor locally, for each subsystem i we need not only its plantified specification $\mathbf{K}_i$, but also a *complementary plantified specification* concerning the rest of the plant from the perspective of i. So, let us redefine the contract as a single automaton $\mathbf{C}_i = \mathbf{A}_i \parallel \mathbf{G}_i$, which then is the complementary plantified specification we need – since $\mathbf{A}_i$ models the complement of the controlled plant, and $\mathbf{G}_i$ the specification that arises from this complement due to cooperation.

Contract Compliance. For the local specifications and also the guarantee to be satisfied, the contracts should be

obtained in such a way that a local supervisor does not assume more restrictions are being imposed by the other closed-loop subsystem than the latter actually guarantees. Therefore, their contracts should be *compatible*, meaning $\mathcal{L}(\mathbf{G}_j) \subseteq \mathcal{L}(\mathbf{A}_i)$ and $G_j \subseteq A_i$. In addition, since we want to avoid overly restrictive local supervisors, the guarantees should be as permissive as possible, i.e., $\mathcal{L}(\mathbf{G}_j) = \mathcal{L}(\mathbf{A}_i)$ and $G_j = A_i$. This implies that the contract automata $\mathbf{C}_1$ and $\mathbf{C}_2$ should be *equivalent*. Our goal is then to find automata $\mathbf{S}_i$ that allow us to define the desired supervisors f_i, and that are *compliant* with equivalent contracts. An automaton $\mathbf{S}_i$ is compliant with a contract $\mathbf{C}$ over Σ^{s} if $P_{i\mathrm{s}}(\mathcal{L}(\mathbf{S}_i)) = \mathcal{L}(\mathbf{C})$ and $P_{i\mathrm{s}}(S_i) = C$.

Outline. In Sec. IV, we introduce a procedure for contract negotiation that computes automata $\mathbf{S}_i$ compliant with equivalent contracts $\mathbf{C}_i$, while assuming cooperation and respecting privacy. These automata may need to be further refined by auxiliary functions introduced and discussed in Sec. IV-D and V. Finally, in Sec. VI, we present our *Contract-Based Supervisor Synthesis* procedure, which combines contract negotiation with the auxiliary functions in order to implement local supervisors imposing a nonblocking global behaviour and satisfying the local specifications. The proofs of all the results presented in this paper can be found in Appendix I.

IV. Contract Negotiation

This section introduces a negotiation framework to compute supervisors f_i for each subsystem in a *decentralised* manner and while respecting *privacy*. These supervisors are maximally permissive if certain sufficient local conditions, which we establish in the current and in the next section, are satisfied. The negotiation is based on a *local* iterative refinement of the plantified specification automata $\mathbf{K}_i$. At each iteration and for each subsystem, states and transitions are removed from $\mathbf{K}_i$ to prevent blocking, ensure controllability, and to satisfy the guarantees $\mathbf{G}_i$ for the other subsystem, while relying on the latter to enforce the assumptions $\mathbf{A}_i$. The negotiation takes place in a way that avoids the plant to be overly restricted. Thus, if the aforementioned conditions can be guaranteed after the first fixed point of negotiation is found, the resulting supervisors are maximally permissive. Otherwise, as we discuss in this and the next sections, the global closed-loop behaviour may still be blocking, which can be both tested and fixed fully locally, even though this may cost the maximally permissive trait in order to protect privacy.

A. Cooperative Supervisor Synthesis

Let us recall that, in the setting we consider here, the local systems may not agree on the controllability status of their shared events. Inspired by [9], the idea is that their local supervisors cooperate in such a way as to assist each other in disabling shared events that are uncontrollable for one, but controllable for the other – otherwise, control actions designed locally may over-restrict the behaviour of the plant.

To understand why *cooperative* local supervisor synthesis is needed to preserve maximal permissiveness, note that, for each subsystem $\mathbf{M}_i$, its set of uncontrollable events is $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}} \dot\cup \Sigma^{\mathrm{s}}_{i,\mathrm{cp}} \dot\cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$. If we consider controllability with respect to those events while computing $\textsc{Synth}(\mathbf{K}_i)$, states from $\mathbf{K}_i$ where the only uncontrollable events being disabled are precisely the ones the other system can control are eliminated, since they disrespect the controllability property. However, theses states would not be removed in the monolithic approach, unless they were blocking or unreachable states; the reason is that events controllable either by one or by the other subsystem would be considered as controllable by the global plant. To illustrate this, consider a trivial example where both subsystems have exactly the same behaviour and already respect their specification – that is, $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{K}_1$ and $\mathbf{K}_2$ all have the same semantics. Both subsystems have full knowledge about each other without further exchange of contracts, as all events are shared and controllable by some subsystem. Even so, without cooperation it may not be possible to locally design maximally permissive supervisors, as depicted in Figure 2.

Fig. 2. Lack of cooperation may cause over-restrictive behaviour. Consider that $\boldsymbol{b}, \boldsymbol{e} \in \boldsymbol{\Sigma^{\mathrm{s}}_{1,\mathrm{c}} \backslash \Sigma^{\mathrm{s}}_{2,\mathrm{c}}}$ and $\boldsymbol{a}, \boldsymbol{f} \in \boldsymbol{\Sigma^{\mathrm{s}}_{2,\mathrm{c}} \backslash \Sigma^{\mathrm{s}}_{1,\mathrm{c}}}$; then, the elimination of blocking states in $\mathbf{K_{1,2}}$, as depicted in gray colour, generates a controllability problem in different states for each subsystem, which in turn have to be removed as well; thus, the languages $\sup\boldsymbol{\mathcal{C}(K_i)}$ – recognised by the automata $\mathbf{S}_i$ – are conflicting. If cooperation was assumed, states 1 and 2 would be preserved, and $\mathbf{S_1} \parallel \mathbf{S_2}$ would be equivalent to the automaton $\mathbf{K_{1,2}}$ apart from its blocking states in gray.

Formally, this cooperation requires to perform $\textsc{Synth}$ of $\mathbf{K}_i$ with respect to the uncontrollable event set $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}} \dot\cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$, instead of the actual locally uncontrollable event set $\Sigma_{i,\mathrm{uc}} = \Sigma^{\mathrm{p}}_{i,\mathrm{uc}} \dot\cup \Sigma^{\mathrm{s}}_{i,\mathrm{cp}} \dot\cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$. However, even so, we could still be imposing a more restricted behaviour than needed if the plant alphabet contains shared events no one controls, namely, if $\Sigma^{\mathrm{s}}_{\mathrm{uc}}$ is not empty. The reason is that, in order to prevent blocking, we may try to locally disable a transition by such an event that is not even possible in the global behaviour of the plant; this would needlessly cause the disabling of other transitions to deter the first from happening, thus over-restricting the behaviour of the plant, as depicted in Figure 3.

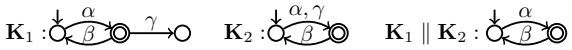

Fig. 3. Consider these two plantified specifications $\mathbf{K}_i$, with alphabets $\boldsymbol{\Sigma_i = \Sigma^{\mathrm{s}} = \{\alpha, \beta, \gamma\}}$. Note that if $\gamma \in \boldsymbol{\Sigma^{\mathrm{s}}_{\mathrm{uc}}}$, the semantics of $\textsc{Synth}(\mathbf{K_1})$ are empty if we compute it assuming $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ as controllable, and $\boldsymbol{\gamma}$ as uncontrollable. However, since the sequence of events $\boldsymbol{\alpha\gamma}$ is not in $\boldsymbol{\mathcal{L}(\mathbf{K_2})}$, it is also not possible in the (nonblocking) global behaviour $\boldsymbol{\mathcal{L}(\mathbf{K_1} \parallel \mathbf{K_2})}$.

To avoid this kind of situation, we apply *halfway synthesis* [13], postponing the decision of eliminating such transitions to after the negotiation of contracts reaches a fixed point. For that end, we state below the definition of the *cooperative* and *partial* synthesis function – adapted from [13].

Definition 2: Take an automaton $\mathbf{S}^0_i$ and a trim automaton $\mathbf{\Lambda}_i = (Q, \Sigma, \delta, q_0, Q_{\mathrm{m}})$ that accepts the language $\sup\mathcal{C}(S^0_i)$ with respect to $\mathbf{S}^0_i$ and $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}}$. Then the *partial cooperative* synthesis function is defined as $\textsc{PCSynth}(\mathbf{S}^0_i) = (Q \cup \{\bot\}, \Sigma, \delta', q_0, Q_{\mathrm{m}})$, where $\bot \notin Q$, and $\delta' = \delta \cup \{(x, \sigma, \bot) \,|\, \exists s \in \Sigma^*_i : x = \delta(q_0, s), \sigma \in \Sigma^{\mathrm{s}}_{\mathrm{uc}}, s\sigma \in \mathcal{L}(\mathbf{S}^0_i)$ and $\delta(x, \sigma)$ is undefined$\}$. $\square$

Note that $\text{PCSYNTH}(\mathbf{S}_i^0)$ is nonblocking apart from a single blocking state, namely $\perp$, reachable only by events in $\Sigma_{\text{uc}}^{\text{s}}$ from states that have these events enabled in $\mathbf{S}_i^0$.

Remark 2: If $\mathbf{K}_i$ is a plantified specification for the subsystem $\mathbf{M}_i$, and $\mathbf{S}_i = \text{PCSYNTH}(\mathbf{K}_i)$, then we have that $S_i = \sup\mathcal{C}(K_i)$ with respect to $\mathcal{L}(\mathbf{M}_i)$ and $\Sigma_{i,\text{uc}}^{\text{p}}$. Moreover, if we take controllability with respect to $\mathcal{L}(\mathbf{M}_i)$ and $\Sigma_{i,\text{uc}}^{\text{p}} \cup \Sigma_{\text{uc}}^{\text{s}}$, then $\sup\mathcal{C}(S_i) = \sup\mathcal{C}(K_i)$. □

B. Contract Extraction

Consider the automaton $\mathbf{S}_i := \text{PCSYNTH}(\mathbf{K}_i)$. It implicitly disables shared events that are not controllable in $\mathbf{M}_i$ but are in $\mathbf{M}_j$, presuming that the supervisor of the latter will assist that of the former in disabling them. At the same time, $\mathbf{S}_i$ also disables locally controllable events to achieve its own local specifications, ensure controllability with respect to $\Sigma_{i,\text{uc}}^{\text{p}}$ and local nonblockingness apart from the state $\perp$ from Def. 2. This, in turn, may also restrict the occurrence of shared events in $\mathbf{S}_j$ due to the synchronisation in the global behaviour.

Motivated by the discussion in Sec. III-B, a natural choice for the first draft of a contract automaton $\mathbf{C}_i$ – that is generated by $\mathbf{M}_j$ to be used by $\mathbf{M}_i$ during the next round of negotiation – is the *observer* automaton of $\mathbf{S}_j$ over Σ^{s}, defined as follows.

Definition 3 (Observer [17]): For an automaton $\mathbf{\Lambda} = (Q, \Sigma, \delta, q_0, Q_{\text{m}})$, with $\Sigma = \Sigma^{\text{s}} \dot\cup \Sigma^{\text{p}}$, the *unobservable reach* of a state $q \in Q$ is defined as $UR(q) = \{\hat{q} \in Q \mid \exists w \in \Sigma^{\text{p}*} : \delta(q, w) = \hat{q}\}$. It can be extended to sets $X \subseteq Q$ by $UR(X) = \cup_{x \in X} UR(x)$. Analogously, we define the observable reach of X by an event $\sigma \in \Sigma^{\text{s}}$ as $OR(X, \sigma) = \{q \in Q \mid \exists x \in X : \delta(x, \sigma) = q\}$. The *observer* $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\text{s}}} := (Q^{\text{o}}, \Sigma^{\text{s}}, \delta^{\text{o}}, q_0^{\text{o}}, Q_{\text{m}}^{\text{o}})$ can then be constructed iteratively as follows, until no more reachable states can be added to Q^{o}.

1) Define the initial state as $q_0^{\text{o}} = UR(q_0)$.
2) For any $q^{\text{o}} \in Q^{\text{o}}$ and for any $\sigma \in \Sigma^{\text{s}}$, take $\hat{q}^{\text{o}} = UR\big(OR(q^{\text{o}}, \sigma)\big)$; if $\hat{q}^{\text{o}} \neq \emptyset$, add $\hat{q}^{\text{o}}$ to Q^{o}, and define $\delta^{\text{o}}(q^{\text{o}}, \sigma) = \hat{q}^{\text{o}}$.

Then $Q_{\text{m}}^{\text{o}} = \{q^{\text{o}} \in Q^{\text{o}} \mid \exists q \in q^{\text{o}} : q \in Q_{\text{m}}\}$. □

The procedure to compute an observer and its properties are described in [17]. By defining $\mathbf{C}_i := \mathbf{O}_{\mathbf{S}_j,\Sigma^{\text{s}}}$, the contract generates and accepts the languages $P_{j\text{s}}(\mathcal{L}(\mathbf{S}_j))$ and $P_{j\text{s}}(S_j)$.

C. Negotiation

Based on the foregoing discussion, we propose the following iterative procedure for the negotiation of contracts. Initially, we compute $\mathbf{S}_i^1 := \text{PCSYNTH}(\mathbf{K}_i)$ and extract the first contract drafts $\mathbf{C}_i^1$. Next, we compute new supervisors restricted to the latest drafts of contracts, namely, $\mathbf{S}_i^1 \parallel \mathbf{C}_i^1$. By taking this composition, we may remove some or even all transitions to the blocking state $\perp$ introduced by PCSYNTH; however, this may also introduce new controllability or blocking problems. For this reason, we perform PCSYNTH again. Inducing this argument for an arbitrary step $k > 1$, we have that $\mathbf{S}_i^k := \text{PCSYNTH}(\mathbf{S}_i^{k-1} \parallel \mathbf{C}_i^{k-1})$, which generates a new draft of contract given by $\mathbf{C}_i^k := \mathbf{O}_{\mathbf{S}_j^k,\Sigma^{\text{s}}}$.

Procedure 1 NEGOTIATION

Require: Automata $\mathbf{S}_1^0$ and $\mathbf{S}_2^0$
$\mathbf{S}_1 \leftarrow \text{PCSYNTH}(\mathbf{S}_1^0)$ and $\mathbf{S}_2 \leftarrow \text{PCSYNTH}(\mathbf{S}_2^0)$
$\mathbf{C}_1 \leftarrow \mathbf{O}_{\mathbf{S}_2,\Sigma^{\text{s}}}$ and $\mathbf{C}_2 \leftarrow \mathbf{O}_{\mathbf{S}_1,\Sigma^{\text{s}}}$
cond ← `True`
while *cond* **do**
  $\mathbf{S}_1' \leftarrow \mathbf{S}_1 \parallel \mathbf{C}_1$ and $\mathbf{S}_2' \leftarrow \mathbf{S}_2 \parallel \mathbf{C}_2$
  if $\mathbf{S}_1 \not\equiv \mathbf{S}_1'$ or $\mathbf{S}_2 \not\equiv \mathbf{S}_2'$ **then**
    for $i \in \{1, 2\}$ **do**
      if $\mathbf{S}_i \not\equiv \mathbf{S}_i'$ **then**
        $\mathbf{S}_i \leftarrow \text{PCSYNTH}(\mathbf{S}_i')$ and $\mathbf{C}_j \leftarrow \mathbf{O}_{\mathbf{S}_i,\Sigma^{\text{s}}}$ $(j \neq i)$
  else *cond* ← `False`
return $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{C}_1, \mathbf{C}_2)$.

The entire procedure, called NEGOTIATION, is detailed in Proc. 1. It iteratively refines $\mathbf{S}_i$ and $\mathbf{C}_i$ until $\mathbf{S}_i^k \equiv \mathbf{S}_i^k \parallel \mathbf{C}_i^k$, for some $k \geq 1$, implying that no new controllability problem with respect to $\Sigma_{i,\text{uc}}^{\text{p}}$ or blockingness apart of $\perp$ need to be solved. Later on, in Sec. VI, we will use as inputs for this procedure automata that are different from $\mathbf{K}_i$. Therefore, to be more precise, let us define the condition that any pair of automata given as inputs for NEGOTIATION should satisfy in order for the subsequent lemma to hold.

Condition 1: Consider a plant composed of two subsystems $\mathbf{M}_i$ over Σ_i, as described in Sec. III-A. Then, for any pair of automata $\mathbf{\Lambda}_i$ over Σ_i, we define the following requirements:

1) $\mathcal{L}(\mathbf{\Lambda}_i) \subseteq \mathcal{L}(\mathbf{M}_i)$ and $\Lambda_i \subseteq M_i$;
2) $(\forall s \in \overline{\Lambda_i}, \forall \sigma \in \Sigma_{i,\text{uc}}^{\text{p}})\ \big(s\sigma \notin \overline{\Lambda_i}$ and $s\sigma \in \mathcal{L}(\mathbf{M}_i)\big) \rightarrow s\sigma \in \mathcal{L}(\mathbf{\Lambda}_i)$;
3) $(\forall s \in \overline{\Lambda_i}, \forall s' \in \overline{\Lambda_j}, \forall \sigma \in \Sigma_{\text{uc}}^{\text{s}})\ \big(P_{j\text{s}}(s') = P_{i\text{s}}(s), s\sigma \in \mathcal{L}(\mathbf{M}_i), s'\sigma \in \mathcal{L}(\mathbf{M}_j)$ and $s\sigma \notin \overline{\Lambda_i}\big) \rightarrow s\sigma \in \mathcal{L}(\mathbf{\Lambda}_i)$ □.

Lemma 1: Let $\mathbf{S}_i^0$ be any pair of automata that satisfies Cond. 1, and $\mathbf{S}_i$ be the outputs of Proc. 1 with $\mathbf{S}_i^0$ as its inputs, namely, $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{C}_1, \mathbf{C}_2) = \text{NEGOTIATION}(\mathbf{S}_1^0, \mathbf{S}_2^0)$. Then the pair $\mathbf{S}_i$ also satisfies Cond. 1, and both automata are compliant with the contracts $\mathbf{C}_i$, which are equivalent. □

Remark 3: If $\mathbf{K}_i$ are plantified specification for the subsystems $\mathbf{M}_i$, then the pair $(\mathbf{K}_1, \mathbf{K}_2)$ satisfies Cond. 1. □

Finally we have that this procedure is not overly restrictive, as stated in the following theorem. The iterative refinements of the inputs by NEGOTIATION do not remove any more states and transitions than the necessary to obtain $\sup\mathcal{C}(S_1^0 \parallel S_2^0)$. They may remove less, though, in which case we further need to guarantee that the resulting composed system is nonblocking, as we will discuss from this point on.

Theorem 1: In the context of Lem. 1, we have that $\sup\mathcal{C}(S_1 \parallel S_2) = \sup\mathcal{C}(S_1^0 \parallel S_2^0)$, where $\sup\mathcal{C}$ is taken with respect to $\mathcal{L}(\mathbf{M}_1 \parallel \mathbf{M}_2)$ and $\Sigma_{1,\text{uc}}^{\text{p}} \cup \Sigma_{2,\text{uc}}^{\text{p}} \cup \Sigma_{\text{uc}}^{\text{s}}$. □

Corollary 1: Given the premisses of Lem. 1, if $\mathbf{S}_i$ are trim automata and S_1 and S_2 are nonconflicting, we have that $S_1 \parallel S_2 = \sup\mathcal{C}(S_1^0 \parallel S_2^0)$, where $\sup\mathcal{C}$ is taken with respect to $\mathcal{L}(\mathbf{M}_1 \parallel \mathbf{M}_2)$ and $\Sigma_{1,\text{uc}}^{\text{p}} \cup \Sigma_{2,\text{uc}}^{\text{p}} \cup \Sigma_{\text{uc}}^{\text{s}}$. □

D. Treating the residual blocking state

Notice that, because of Def. 2, it is not always the case that $\mathbf{S}_i$ are trim – except if all shared events are controllable by at least one subsystem, i.e., if $\Sigma_{\text{uc}}^{\text{s}}$ is empty, as we considered in [16]. If NEGOTIATION converges to a fixed point where a blocking state $\perp$ is preserved on at least one of the automata

$\mathbf{S}_i$, we need to disable each one of the transitions by shared uncontrollable events leading to a global blocking state caused by a local $\perp$. To help us tackle this problem and make it more precise, consider the definition below.

Definition 4: Given the premisses of Lem. 1, and for $\mathbf{S}_i = (Q_i, \Sigma_i, \delta_i, q_{i_0}, Q_{i\mathrm{m}})$ we denote by *residuals* of $\mathbf{S}_i$ the set

$$R_i = \big\{(s^i, \sigma) \in \mathcal{L}(\mathbf{S}_i) \times \Sigma^{\mathrm{s}}_{\mathrm{uc}} \mid \delta_i(q_{i_0}, s^i\sigma) = \perp\big\}.$$

Then, for each i and each residual $r = (s^i, \sigma) \in R_i$, we define $\mathit{illegal}_i(r)$ as the state x such that $x = \delta_i(q_{i_0}, s^i)$, and the set $\mathit{risky}_j(r) = \big\{y \in Q_j \mid \exists s^j \in \Sigma_j^* \,.\, \delta_j(q_{j_0}, s^j) = y, P_{j\mathrm{s}}(s^j) = P_{i\mathrm{s}}(s^i), \text{ and } \delta_j(y, \sigma)!\big\}$. □

The set R_i describes all the problematic local words that lead to the state $\perp$ left in $\mathbf{S}_i$ by negotiation. The *illegal* of a residual $r \in R_i$ is a state that precedes $\perp$ in $\mathbf{S}_i$. The *risky* set contains all the states in $\mathbf{S}_j$ that synchronise – in the global behaviour – with the corresponding illegal state in $\mathbf{S}_i$, and that enable the shared uncontrollable event that leads to $\perp$ in $\mathbf{S}_i$.

For each residual $r \in R_i$ we need – and have the flexibility – to choose whether to remove $\mathit{illegal}_i(r)$ from $\mathbf{S}_i$, or all the states in $\mathit{risky}_j(r)$ from $\mathbf{S}_j$. However, this decision affects the permissiveness of the global controlled behaviour in a way that is not obvious from a local perspective. Moreover, there are cases where, no matter which choice is taken, the result is not equivalent to the one obtained with the monolithic approach.

In Sec. VI we will see that the consequences of this decision are hard to predict especially because NEGOTIATION and the procedure that treats the residuals – called TREATRESIDUAL and defined in the following – are embedded in a loop in our synthesis framework – see Fig. 9 and Fig. 10. This framework explores different choices, not only about where to get rid of the residuals from negotiation, but also about other decisions we will face in Sec. VI. In principle, our framework could try all the described possibilities. However, to avoid combinatorial explosion, we consider that all the residuals from both R_1 and R_2 are treated in the same automaton $\mathbf{S}_i$, for a chosen i.

With that in mind, we define TREATRESIDUAL in Proc. 2. It gets as inputs the automaton $\mathbf{S}_i$ chosen to treat all the residuals from both R_1 and R_2, and both contracts $\mathbf{C}_i$ outputted by NEGOTIATION. As we point out in the remark below, these contracts are useful to determine which states should be removed. To this end, we use the synchronous composition of $\mathbf{C}_1$ and $\mathbf{C}_2$ to get a single automaton $\mathbf{C}$, since it is likely that the contracts, although equivalent, are not represented by the same automaton. From $\mathbf{C}$, we can determine which states should be removed to treat R_1 and R_2; that is, we can find the state $\mathit{illegal}_i(r_i)$ for all the residuals r_i in R_i (line 5), and the states $\mathit{risky}_i(r_j)$ for all the residuals r_j in R_j (line 7).

Remark 4: For all states $q = (q_1, q_2)$ and $q' = (q'_1, q'_2)$ in $\mathbf{C}$ with transition function δ, for all $\sigma \in \Sigma^{\mathrm{s}}_{\mathrm{uc}}$ such that $\delta(q, \sigma) = q'$, and for all i, we have that $\perp \in q'_j$ iff there is $r \in R_i$ where $\mathit{illegal}_i(r) \in q_j$ and $\mathit{risky}_j(r) \subseteq q_i$ (note that q_i, q'_i are states in the observer of $\mathbf{S}_j$, and q_j, q'_j of $\mathbf{S}_i$). □

Note that TREATRESIDUAL will not return a trim automaton; this procedure only solves the residuals, but leaves controllability and possibly new blocking problems to be solved by PCSYNTH in NEGOTIATION. Moreover, TREATRESIDUAL may cause the loss of the fixed point properties of NEGOTIATION – see Lem. 1 – so we would need to execute the latter again, regardless. We discuss more about this in Sec. VI.

Procedure 2 TREATRESIDUAL

Require: Automata $\mathbf{S}_i$, $\mathbf{C}_1$ and $\mathbf{C}_2$ where $\mathbf{O}_{\mathbf{S}_i,\Sigma^{\mathrm{s}}} = \mathbf{C}_j \equiv \mathbf{C}_i$
Notation: $\mathbf{\Lambda}.\delta$ and $\mathbf{\Lambda}.Q$ denote, respectively, the transition function and the set of states for any automaton $\mathbf{\Lambda}$

1: $\mathbf{C} \leftarrow \mathbf{C}_1 \parallel \mathbf{C}_2$ and $\mathbf{S} \leftarrow \mathbf{S}_i$
2: $\mathit{fix} \leftarrow \Big\{\big((q_1, q_2), \sigma, (q'_1, q'_2)\big) \in \mathbf{C}.\delta \;\Big|\; \perp \in q'_1 \cup q'_2 \text{ and } \sigma \in \Sigma^{\mathrm{s}}_{\mathrm{uc}}\Big\}$
3: $\mathit{aux} \leftarrow \emptyset$
4: **for** $\big((q_1, q_2), \sigma, (q'_1, q'_2)\big) \in \mathit{fix}$ **do**
5: **if** $\perp \in q'_j$ **then**
6: $\mathit{aux} \leftarrow \mathit{aux} \cup \big\{(x, \sigma, \perp) \in \mathbf{S}.\delta \mid x \in q_j\big\}$
7: **else**
8: $\mathit{aux} \leftarrow \mathit{aux} \cup \big\{(x, \sigma, x') \in \mathbf{S}.\delta \mid x \in q_j \text{ and } x' \in q'_j\big\}$
9: **for** $(x, \alpha, x') \in \mathit{aux}$ **do**
10: **if** $\perp \notin \mathbf{S}.Q$ **then** $\mathbf{S}.Q \leftarrow \mathbf{S}.Q \cup \{\perp\}$
11: $\mathit{pre}_x \leftarrow \big\{(p, \alpha) \in \mathbf{S}.Q \times \Sigma_i \mid (p, \alpha, x) \in \mathbf{S}.\delta\big\}$
12: **for** $(p, \alpha) \in \mathit{pre}_x$ **do**
13: $\mathbf{S}.\delta \leftarrow \Big(\mathbf{S}.\delta \backslash \big\{(p, \alpha, x)\big\}\Big) \cup \big\{(p, \alpha, \perp)\big\}$
14: $\mathbf{S}.\delta \leftarrow \mathbf{S}.\delta \backslash \Big(\big\{(x, \alpha, x')\big\} \cup \big\{(x, \sigma, x'') \mid \sigma \in \Sigma_i \text{ and } x'' \in \mathbf{S}.Q\big\}\Big)$
15: $\mathbf{S}.Q \leftarrow \mathbf{S}.Q \backslash \{x\}$ and remove unreachable states from $\mathbf{S}$
16: **return** $\mathbf{S}$

Remark 5: If a pair $(\mathbf{S}^0_1, \mathbf{S}^0_2)$ satisfies Cond. 1, and given equivalent contracts $\mathbf{C}_i = \mathbf{O}_{\mathbf{S}^0_j,\Sigma^{\mathrm{s}}}$, then $(\mathbf{S}_i, \mathbf{S}^0_j)$ also satisfies Cond. 1, for $\mathbf{S}_i = \mathrm{TREATRESIDUAL}(\mathbf{S}^0_i, \mathbf{C}_1, \mathbf{C}_2)$. □

E. Extraction of Cooperative Supervisors

From this point on, until we start Sec. VI, let us assume $\mathbf{S}_i$ returned by NEGOTIATION are trim automata. As they are computed assuming *cooperation*, by Def. 2 (PCSYNTH) and by Rem. 2, we can guarantee that local controllability of S_i is respected in terms of $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$, but not in terms of $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$. That is, as we allow the processes to have different controllability settings over certain shared events – namely, in $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$ – even though events in $\Sigma^{\mathrm{s}}_{\mathrm{c}} \supset \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$ are considered globally controllable, they are not locally controllable. Therefore, even if $\mathbf{S}_i$ are trim, we cannot combine local control actions f_i such that $\mathbf{S}_i \sim f_i$, as in Sec. II-B.

To illustrate that, let us say that, in order to respect local controllability, we extract a local supervisor $f_i : \mathcal{L}(\mathbf{M}_i) \to \Gamma_i$ which only disables locally controllable events in the set $\Sigma_{i,\mathrm{c}}$, that is, for all $s \in \mathcal{L}(\mathbf{M}_i)$, define $f_i(s) = \{\sigma \in \Sigma_i \mid s\sigma \in \overline{S_i}\} \cup \Sigma^{\mathrm{s}}_{i,\mathrm{cp}} \subseteq \Sigma_i$. In this case, the simple example depicted in Fig. 4 illustrates why cooperation may fail: supervisor i expects cooperation from j to disable, at the state 0, the event $\theta \in \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$; however, this is not expressed in the contract, because i needs θ to be enabled at state 1, while states 0 and 1 correspond to the same state in the contract, and therefore are indistinguishable to j.

Fig. 4. Scenario where cooperation fails. The event $\boldsymbol{\alpha}$ is local, while $\boldsymbol{\theta}$ is shared but locally uncontrollable for $\mathbf{S}_i$, while controllable for $\mathbf{S}_j$; the transition with $\boldsymbol{\theta}$ from state 0 needs to be disabled and $\boldsymbol{f_i}$ cannot do this, so cooperation is needed; however, since a transition by the same event is enabled in state 1, it is not possible to represent that disablement in the observer of $\mathbf{S}_i$, i.e., in contract $\mathbf{C}_j$.

To solve this problem, we propose that the supervisors cooperate in achieving joint control ([18]) by the use of a

signalling bit for each shared event σ in $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$. The idea is that supervisor i, which does not control σ, sets the corresponding bit to one as a request for supervisor j, which controls σ, to disable this event on i's behalf. The value of these bits related to $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$ are written only by supervisor i, but can be read by supervisor j. We describe their behaviour by a function $d_i : \mathcal{L}(\mathbf{M}_i) \to 2^{\Sigma^{\mathrm{d}}_i}$, where $\mathrm{d}^\sigma \in \Sigma^{\mathrm{d}}_i$, as defined in Sec. III-A, represents a request to disable σ. For all $w_i \in \mathcal{L}(\mathbf{M}_i)$ and all $\mathrm{d}^\sigma \in \Sigma^{\mathrm{d}}_i$, we interpret this function such that $\mathrm{d}^\sigma \in d_i(w_i)$ if and only if supervisor i assigns one to the bit relative to $\sigma \in \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$ after observing w_i. We then define $d_i(w_i) := \{\mathrm{d}^\sigma \mid w_i\sigma \notin \overline{S_i}\}$. We assume that the signalling bits are instantaneously updated by supervisor i with the occurrence of a local event. That is, we assume, for all $\alpha \in \Sigma_i$, that the output $d_i = d_i\big(w_i\big)$ is immediately updated to $d_i = d_i\big(w_i\alpha\big)$ with the occurrence of α. This allows supervisor i to inform j its control decision with respect to the shared events in $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$. We then extract supervisors as follows, which allows us to state Lem. 2 below.

Definition 5: Given the premisses of Lem. 1, and if $\mathbf{S}_i$ are nonempty trim automata, we define each local supervisor $f_i : \mathcal{L}(\mathbf{M}_i) \times 2^{\Sigma^{\mathrm{d}}_j} \to \Gamma_i$, with $\Gamma_i \subseteq 2^{\Sigma_i}$, such that for all $w_i \in \mathcal{L}(\mathbf{M}_i)$ and for all $d_j \in 2^{\Sigma^{\mathrm{d}}_j}$, we have $f_i(w_i, d_j) = \{\sigma \mid w_i\sigma \in \overline{S_i}$ and $(\sigma \in \Sigma^{\mathrm{s}}_{j,\mathrm{cp}} \to \mathrm{d}^\sigma \notin d_j)\} \cup \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$.

Lemma 2: In the context of Def. 5, we have that

1) $\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2) = S_1 \parallel S_2$ and
2) $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{S_1} \parallel \overline{S_2}$. □

From this lemma, the next result immediately holds.

Corollary 2: The global behaviour of the plant in closed loop with the supervisors f_i from Def. 5 is nonblocking, i.e.,

$$\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2)},$$

if and only if S_1 and S_2 are nonconflicting. □

Remark 6: Related to the requests to disable events in $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$ through the use of signalling bits, as described above, we highlight the following points. First, note that if supervisors agree on the controllability status of shared events – which is typically assumed in the literature – then $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}} = \emptyset$ for all i, and no signalling bit, nor the function d_i, are needed. Second, when that is not the case, then this sort of communication is necessary only for the *implementation* of the supervisor functions f_i, and not for the synthesis of $\mathbf{S}_i$ – as opposed to the exchange of contracts during NEGOTIATION. The signalling bits represented by the function d_i simply guarantee that the supervisor j will enforce, with respect to $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$, the joint control action obtained from $\mathbf{S}_1$ and $\mathbf{S}_2$.

Third, what we propose here is different from communicating private events, as e.g. in [10], where a subset of the exclusively local (private) events needs to be observed by other supervisors – i.e., every occurrence of such events is communicated. Here, a request d^σ just informs a supervisor j that a shared event σ controlled only by j needs to be disabled, even when the contract says otherwise. While indirectly this might mean some private event occurred in subsystem i, supervisor j is not informed which one it was, let alone its every occurrence. Supervisor j is also not informed about how many private events occurred in i before the value of the signalling bits changed, i.e., j observes the output of the function d_i, but not its input. Moreover, in principle supervisor j cannot even infer this information, as our approach does not share the local model of each subsystem – apart from the contract that is only described in terms of the shared events – neither during the synthesis nor the execution of the supervisors. Finally, even if $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}} \neq \emptyset$, we can still avoid the signalling bits and the function d_i by using the definition of *relative unambiguity* from [16], instead of redefining it as a weaker property in Def. 7 – see Rem. 8. □

V. ASSURING NONCONFLICTING LOCAL SUPERVISORS

Assume in this section again that the supervisor automata $\mathbf{S}_i$ resulting from negotiation are trim; in the next section, we will treat the general case. This section shows how these local supervisors, although locally nonblocking, may have to be further restricted to ensure a nonblocking behaviour of the resulting global closed-loop system. As we aim at computing necessary restrictions fully locally to preserve privacy, we might need to sacrifice maximal permissiveness.

A. Local Property for Nonconflict: A Motivation

Consider the example in Fig. 5, where $\mathbf{S}_1$ and $\mathbf{S}_2$ are trim automata with shared alphabet $\Sigma^{\mathrm{s}} = \{\alpha, \beta, \theta\}$; assume controllability (with cooperation) is not being disrespected. They satisfy the fixed point property for NEGOTIATION; however, their composed behaviour is blocking.

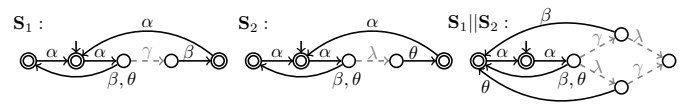

Fig. 5. Conflicting supervisors computed by Proc. 1.

To understand the reason behind this conflict, note that the strings α and $\alpha\lambda$ in $\overline{S_2}$ look the same to $\mathbf{S}_1$, as $P_{2\mathrm{s}}(\alpha) = P_{2\mathrm{s}}(\alpha\lambda) = \alpha$. This conceals the fact that β can occur in $\mathbf{S}_2$ after α, but not after $\alpha\lambda$. Analogously, α and $\alpha\gamma$ look the same to $\mathbf{S}_2$, hiding the fact that θ can occur in $\overline{S_1}$ after α, but not after $\alpha\gamma$. Thus, strings like $\alpha\gamma\lambda$ or $\alpha\lambda\gamma$ can occur in $\overline{S_1} \parallel \overline{S_2}$ and lead to blocking states.

B. The Unambiguity Property

Consider the global behaviour of the plant whose subsystems are in closed loop with their respective local supervisor, where these supervisors were obtained from negotiation, i.e., are compliant with equivalent contracts. To ensure nonconflict, or global nonblockingness, in this system – preventing situations as the one just described in the last subsection – it is sufficient to guarantee that each controlled subsystem satisfies the following property, here named *unambiguity*.

Definition 6: The language S_i is *unambiguous* with respect to Σ^{s} and the natural projection $P_{i\mathrm{s}} : \Sigma_i^* \to \Sigma^{\mathrm{s}*}$ if, for any $s, s' \in \overline{S_i}$ such that $P_{i\mathrm{s}}(s) = P_{i\mathrm{s}}(s')$,

1) $(\forall \sigma \in \Sigma^{\mathrm{s}})\ s\sigma \in \overline{S_i} \ \Rightarrow (\exists s'' \in \Sigma_i^*)$
$s' \leq s''$, $P_{i\mathrm{s}}(s'') = P_{i\mathrm{s}}(s')$ and $s''\sigma \in \overline{S_i}$; and
2) $s \in S_i \Rightarrow (\exists s'' \in \Sigma_i^*)$
$s'' \leq s'$or $s' < s''$, $P_{i\mathrm{s}}(s'') = P_{i\mathrm{s}}(s')$ and $s'' \in S_i$. □

The reasons for the name are the following. A string s_s over the shared alphabet Σ^{s} can be the projection of different strings over a local alphabet Σ_i; some may allow a shared event σ to eventually happen, possibly after a suffix string of nonshared events, while others may not; thus, this ambiguity of s_s makes it possible for conflict to happen – e.g., consider as s_s the projection α in the example from Fig. 5. This scenario is prevented by the condition in Def. 6.1, which also appears in [7], where it is called *locally nonblocking* condition.

Now, assume each automaton $\mathbf{S}_i$ satisfies Def. 6.1, but not Def. 6.2; then, their marking is ambiguous and, although all marked states of each automaton can be reached in the composed behaviour, conflict may still arise if for some path they are never reached simultaneously in both automata.

To better explain this notion, let us take a look at the Fig. 6, where $\mathbf{S}_i$ are trim automata, $\Sigma^{\mathrm{s}} = \{\alpha, \beta\}$, and $\mathbf{O}_i = \mathbf{O}_{\mathbf{S}_i,\Sigma^{\mathrm{s}}}$. Assume local controllability is not being disrespected. Let us consider two different markings. In the first one, indicated by the colours black and red, only the states 3 and 5 are marked in $\mathbf{S}_1$, the states $\{1,2,5\}$ and $\{3,4\}$ in $\mathbf{O}_1$, the states 2 and 4 in $\mathbf{S}_2$, and the states $\{1,4\}$ and $\{2,3\}$ in $\mathbf{O}_2$. The second marking is shown by the colours black and blue, so the only marked states in $\mathbf{S}_1$ are 2 and 5, in $\mathbf{O}_1$ the state $\{1,2,5\}$, in $\mathbf{S}_2$, the states 1 and 4, and in $\mathbf{O}_2$, the state $\{1,4\}$.

The supervisors from Fig. 6 satisfy, in both marking scenarios, the fixed point property for NEGOTIATION; moreover, the condition from Def. 6.1 is also satisfied. For the first marking, though, the condition from Def. 6.2 is not. The reason is that from the perspective of one subsystem, the only marking that can be observed from the other one is through its observer. Then, for instance, we have that the sequence α is in the marked language of the observer, while $\mathbf{S}_1$ could be in the state 1 or 2, and neither is marked; moreover, if a β is observed after α, $\mathbf{S}_1$ could be in the state 3 or 4, and again it is not possible to know just by looking at $\mathbf{O}_1$ whether a marked state was actually reached in $\mathbf{S}_1$. This leads to conflict in this example simply because, in the composed behaviour $\mathbf{S}_1 \parallel \mathbf{S}_2$, it is possible to reach states from which the local markings never synchronise. Indeed, in the global behaviour, $\mathbf{S}_2$ may get locked in the loop between the states 3 and 4, while $\mathbf{S}_1$ is locked in the cycle with its states 1, 2 and 3, which indicates that S_i are conflicting. Now consider the example with the second marking. Then, note that there is no conflict and that unambiguity is satisfied – e.g., states 1, 2 and 5 from $\mathbf{S}_1$ correspond to the same state in $\mathbf{O}_1$, and although 1 is the only one not marked, it can reach 2 by local event b.[1]

Based on the intuition from the foregoing discussion, we state the following result.

Proposition 1: For automata $\mathbf{S}_1$ and $\mathbf{S}_2$ as in Lem. 1, if they are trim, and if their marked languages S_1 and S_2 are unambiguous with respect to Σ^{s} and the natural projection $P_{i\mathrm{s}} : \Sigma_i^* \to \Sigma^{\mathrm{s}*}$, then these languages are nonconflicting. □

[1]It is worth mentioning that, although unambiguity might look similar to properties used in hierarchical control to guarantee global nonblockingness, they are not quite the same, as different control architectures and synthesis methods require different conditions to be satisfied. So, for example, although Def. 6.1 is equivalent to *locally nonblocking condition* from [7], we have that Def. 6.2 is not equivalent to *marked string acceptance* from [7] – indeed, for the second marking in Fig. 6, the former is satisfied while the latter is not.

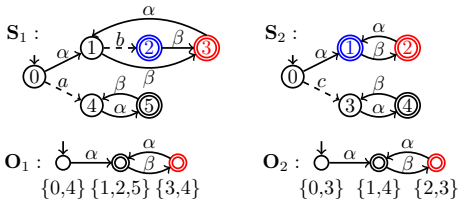

Fig. 6. Example of supervisors computed by Proc. 1. For the first marking considered, they are conflicting, and each $\mathbf{S}_i$ disrespects Def. 6.2. For the second marking they satisfy Def. 6 and are not conflicting.

C. Locally Enforcing Unambiguity

Note that unambiguity is not retained under union of languages and therefore a supremal unambiguous sublanguage of S_i may not exist. Since we do not want to just check whether each subsystem satisfies this property, but rather enforce it over the languages S_i, we need a stronger version of it that is closed under union, so we can obtain a supremal sublanguage of S_i that satisfies a sufficient condition for nonconflict. Inspired by the concept of relative observability from [12], we introduce *relative unambiguity*, where a language S_i is given, relative to which we test whether a sublanguage $S_i' \subseteq S_i$ is unambiguous.

Definition 7: The language $S_i' \subseteq S_i$ is *relatively unambiguous*[2] with respect to S_i and the alphabet Σ^{s} if, for any $s, s' \in \overline{S_i}$ such that $P_{i\mathrm{s}}(s) = P_{i\mathrm{s}}(s')$,

1) $(\forall \sigma \in \Sigma^{\mathrm{s}})\ s\sigma \in \overline{S_i'} \Rightarrow (\exists s'' \in \Sigma_i^*)\ s' \leq s''$ and $P_{i\mathrm{s}}(s'') = P_{i\mathrm{s}}(s')$ and $s''\sigma \in \overline{S_i'}$;
2) $s \in S_i' \Rightarrow (\exists s'' \in \Sigma_i^*)\ (s'' \leq s'$ or $s' < s'')$ and $P_{i\mathrm{s}}(s'') = P_{i\mathrm{s}}(s')$, and $s'' \in S_i'$. □

Procedure 3 ENFORCERU$(\mathbf{\Lambda}^0, \Sigma^{\mathrm{s}})$

Require: Automaton $\mathbf{\Lambda}^0$ over Σ_i, subalphabet Σ^{s}
Notation: $\mathbf{\Lambda}.\delta$, $\mathbf{\Lambda}.Q$ and $\mathbf{\Lambda}.Q_{\mathrm{m}}$ denote, respectively, the transition function, the set of states and the set of marked states for any automaton $\mathbf{\Lambda}$

```
Λ ← Λ^0 and compute O ← O_{Λ,Σ^s}
if ∃q, q' ∈ O.Q, with q ≠ q', such that q ∩ q' ≠ ∅ then
    Λ ← Λ || O and recompute O ← O_{Λ,Σ^s}
M_O ← ∅ and changed ← False
for all q ∈ O.Q_m do
    q_m ← {x ∈ q | x ∈ Λ.Q_m} and q_m? ← q\q_m
    if AmbigMarking(Λ, q_m?, q_m) then M_O ← M_O ∪ {q}
if M_O ≠ ∅ then changed ← True and O.Q_m ← O.Q_m\M_O
repeat
    O ← Trim(O) and Λ ← Λ || O and R_O ← ∅
    for all q ∈ O.Q do
        for all σ ∈ Σ^s such that O.δ(q,σ)! do
            q_σ ← {x ∈ q | Λ.δ(x,σ)!} and q_σ? ← q\q_σ
            if AmbigPath(Λ, q_σ?, q_σ) then
                R_O ← R_O ∪ {(q, σ, O.δ(q,σ))}
    if R_O ≠ ∅ then changed ← True and O.δ ← O.δ\R_O
until R_O = ∅
return Λ and changed
```

In order to enforce relative unambiguity – hence, also unambiguity – we define the function ENFORCERU, in Proc. 3, which manipulates a given automaton $\mathbf{\Lambda}^0$ into an automaton $\mathbf{\Lambda}$ such that Λ is the supremal relatively unambiguous sublanguage of Λ^0, denoted by $\Lambda = \sup\mathcal{RU}(\Lambda^0)$.

To start explaining Proc. 3, first consider an automaton $\mathbf{\Lambda}$ and a natural projection $P : \Sigma^* \to \Sigma^{\mathrm{s}*}$. Let us define the

[2]Def. 7 differs from the property under the same name in [16], see Rem. 8.

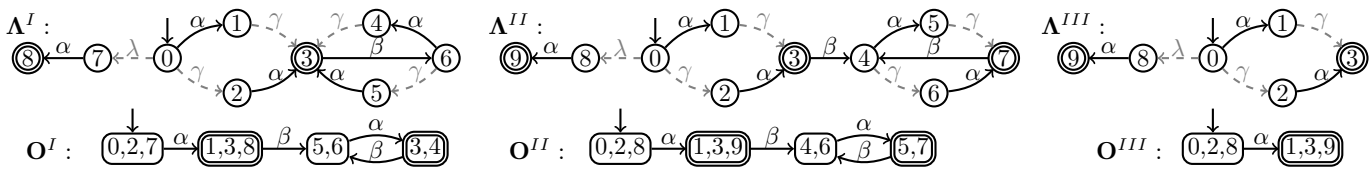

Fig. 7. Illustrative example for Proc. 3 showing automata $\mathbf{\Lambda}$ over $\mathbf{\Sigma}_i$ (top) and $\mathbf{O} = \mathbf{O}_{\mathbf{\Lambda},\mathbf{\Sigma}^{\mathrm{s}}}$ (bottom), where $\mathbf{\Sigma}_i = \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}\}$, and $\mathbf{\Sigma}^{\mathrm{s}} = \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ (transitions by shared and private events represented by solid and dashed arrows, respectively). $\mathbf{\Lambda}^{I}$ does not satisfy Cond. 2 – state **3** appears in more than one state of $\mathbf{O}^{I}$. Then $\mathbf{\Lambda}^{II}$ shows $\mathbf{O}^{I} \parallel \mathbf{\Lambda}^{I}$, where Cond. 2 holds. $\mathbf{\Lambda}^{III}$ shows $\textsc{EnforceRU}(\mathbf{\Lambda}^{I}, \mathbf{\Sigma}^{\mathrm{s}}) = \textsc{EnforceRU}(\mathbf{\Lambda}^{I}, \mathbf{\Sigma}^{\mathrm{s}})$.

uncertainty set of states that a string s in $\mathcal{L}(\mathbf{\Lambda})$ can reach by $U(s) := \{\delta(q_0, s') \,|\, \exists s' \in \mathcal{L}(\mathbf{\Lambda}) : P(s') = P(s)\} \subseteq Q$ [12]. Now, take as $\mathbf{\Lambda}$ the example in Fig. 8, where only the event a is not in the shared alphabet Σ^{s}. The automaton $\mathbf{\Lambda}$ is ambiguous, as the strings $\lambda\zeta$ and $a\lambda\zeta$ look the same under the projection P, but the former can be followed by θ, while the latter cannot. To obtain the automaton that accepts $\sup\mathcal{RU}(\mathbf{\Lambda})$, the event θ must be removed after the string $\lambda\zeta$. However, if we do that in $\mathbf{\Lambda}$, the event θ would be unnecessarily removed after the string $\gamma\alpha$ – and then would have to be removed after $a\gamma\alpha$ as well, overly restricting the behaviour. In order to obtain the supremal sublanguage, we need to adopt for the automaton at hand the same condition as used in [12], [19].

Condition 2: $\big(\forall s, t \in \mathcal{L}(\mathbf{\Lambda})\big)\ \delta(q_0, s) = \delta(q_0, t) \Rightarrow U(s) = U(t)$. □

If this is not satisfied by $\mathbf{\Lambda}$, as in our example, it can be imposed with no loss of generality by replacing $\mathbf{\Lambda}$ with $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\mathrm{s}}} \parallel \mathbf{\Lambda}$ – see [19] for the proof. We guarantee Cond. 2 holds in the if-statement starting in line 2 of Proc. 3.

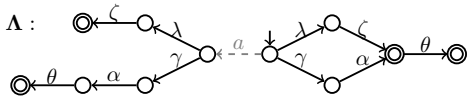

Fig. 8. Example illustrating why Cond. 2 is needed.

The observer $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\mathrm{s}}}$ is not only used for used for Cond. 2, but also informs which states in $\mathbf{\Lambda}$ are reached from the initial state by strings with the same projection over the shared alphabet, which is needed to check relative unambiguity. The first condition that is enforced is the one from Def. 7.2. In the loop starting in line 5 of Proc. 3, we check each marked state q in $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\mathrm{s}}}$. If q contains a nonmarked state from $\mathbf{\Lambda}$ that does not satisfy that condition, q and all the states it contains are respectively unmarked in $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\mathrm{s}}}$ and, through the synchronous product, in $\mathbf{\Lambda}$ (on lines 8 and 10). For such check over the states of q, we call $\textsc{AmbigMarking}$. This function performs back- and forward searches to detect if there is, for each state in the set $q_{\mathrm{m}?}$ (defined on line 6), a sequence of nonshared events that connects this state to any state in the set q_{m} (also defined on line 6), either from the former to the latter, or the other way round; if and only if there is *no* such sequence, the function returns true, i.e., the condition is not satisfied.

To check if the condition from Def. 7.1 holds, in the loop starting in line 11 we inspect each state q in $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\mathrm{s}}}$, which is a set of states in $\mathbf{\Lambda}$. If a shared event σ is enabled from q, there is at least one state in q from which a transition with σ is defined in $\mathbf{\Lambda}$. We can thus partition q into a set q_σ of states in $\mathbf{\Lambda}$ from which σ is enabled, and a set $q_{\sigma?}$ with the remaining states. Then Def. 7.1 is not disrespected if from every state in $q_{\sigma?}$ it is possible to reach (by a sequence of nonshared events) some state in q_σ; this is checked by the function $\textsc{AmbigPath}$ through a backward search from states in q_σ – the function returns true if and only if such reach is *not* possible. If Def. 7.1 is disrespected, the transitions with σ from q and from the states in q_σ need to be removed from $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\mathrm{s}}}$ and $\mathbf{\Lambda}$, respectively, which is done in lines 16 and 10.

As an illustrative example for this procedure, see Fig. 7. It is worth to notice that an automaton resulting from this procedure may not be trim, for we only eliminate from $\mathbf{\Lambda}$ transitions by shared events – that disrespect relative unambiguity – by taking its composition with $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\mathrm{s}}}$. There are two reasons for that. Firstly, transitions by local events can only be removed if we take into account the controllability issues it may bring; thus, this trimming problem is left for the $\textsc{Negotiation}$ function. Secondly, although it is correct to remove states from the observer to enforce the property in question, we cannot remove states from $\mathbf{\Lambda}$ contained in a state of $\mathbf{O}_{\mathbf{\Lambda},\Sigma^{\mathrm{s}}}$ that survived the trimming; such states are needed to remember the original sequences generated by $\mathbf{\Lambda}^0$ – that form the language denoted by $\overline{S_i}$ in Def. 7 – in order to compare them to the sequences generated by the iteratively refined automaton $\mathbf{\Lambda}$ – that form the language denoted by $\overline{S_i'}$ in Def. 7.

Remark 7: Given a pair $(\mathbf{S}_1^0, \mathbf{S}_2^0)$ that satisfies Cond. 1, if $\mathbf{S}_i$ is the output of $\textsc{EnforceRU}(\mathbf{S}_i^0, \Sigma^{\mathrm{s}})$ for some $i \in \{1, 2\}$, then, for $i \neq j$, the pair $(\mathbf{S}_i, \mathbf{S}_j^0)$ also satisfies Cond. 1. □

Remark 8: Please note that the concept of relative unambiguity from Def. 7 differs from the one in [16], as the former requires one less clause than the latter: namely, Def. 7. 3 from [16], which is a relative-observability-like clause with respect to $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$. This clause is obsolete in the present paper because, in order to enable cooperation, the supervisor functions f_i from Def. 5 use signalling bits regarding events in $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$. We consider these bits instead of Def. 7. 3 for the following reasons. Firstly, because this introduces an alternative solution. Note that the one from [16] is also applicable here, despite that we might have $\Sigma^{\mathrm{s}}_{\mathrm{uc}} \neq \emptyset$, while [16] assumes $\Sigma^{\mathrm{s}}_{\mathrm{uc}} = \emptyset$. This is true because: (i) Def. 7. 3 from [16] depends only on $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$, and we have that $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}} \cap \Sigma^{\mathrm{s}}_{\mathrm{uc}} = \emptyset$; (ii) if $\mathbf{S}_i$ are trim, by definition of $\textsc{PCSynth}$, then local controllability of S_i is already respected in terms of $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$ – although not necessarily in terms of $\Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$, reason why either Def. 7. 3 from [16] or the communication proposed here is required.

Secondly, using signalling bits is simpler than enforcing Def. 7. 3 in [16], yet it still retains privacy with respect to Σ^{p}_i, as discussed in Rem. 6. Lastly and most importantly, the use of signalling bits allows for more permissive solutions than if we instead enforce Def. 7 from [16], as this property

is stronger than Def. 7 here. □

VI. Contract-Based Supervisor Synthesis

Our intention in this paper is to compute local supervisors f_i ensuring a nonblocking global closed-loop behaviour that satisfies all given specifications, while respecting the privacy of each subsystem, both during the synthesis of these supervisors, as well as during their execution. With that in mind, we have introduced Proc. 1 in Sec. IV for the negotiation of contracts, which assumes cooperation between the supervisors, and computes automata $\mathbf{S}_i$ that are used to defined these supervisors according to Def. 5. We also have seen that these automata may need further refinements to guarantee global nonblockingness, and for that we defined the auxiliary functions TreatResidual and EnforceRU, in Proc. 2 and 3. Now, we will show how to combine all these procedures in order to fulfil our goal.

Consider the case where some of the outputs of Negotiation are not trim automata. In this case, in principle both the auxiliary functions need to be performed – the first one to indirectly disable transitions that lead to the blocking state $\perp$ added by PCSynth in negotiation, and the other one to check and, if needed, to enforce the property from Def. 7. However, the order in which they are executed affects the resulting supervisor automata $\mathbf{S}_i$ and their permissiveness.

Moreover, the function TreatResidual may remove some transitions that disrespect relative unambiguity, but may also cause new ambiguities. Vice versa, if this property needs to be enforced, EnforceRU may solve some of the current blocking problems left by negotiation, but also introduce new ones. In addition, both functions may cause the loss of the fixed point properties of negotiation – see Lem. 1 – including the compliance with their contracts; so, after calling either of the auxiliary functions, we also need to perform Negotiation again. With these observations we conclude that, after one of the auxiliary functions is performed, both Negotiation and the other auxiliary function need to be called. Again, the question is in which order. As stated in Thm. 1, negotiation is not overly restrictive, while both auxiliary functions can be. Therefore, we give priority to negotiation, that is, we execute Negotiation right after EnforceRU, if the latter actually enforces the property, or right after TreatResidual.

Based on the arguments above, we fix an order in which our procedures can be executed and schematically represent it in Fig. 9. Two more remarks are worth adding about this.

While TreatResidual always modifies its input – i.e., its output is always different from its input – this is not always true for EnforceRU. That is why we need to check the boolean c_i expressing whether the output *changed* with respect to the input $\mathbf{S}_i$. When the input already satisfies the property, it does not need to be enforced, and c_i is false; if c_i is false for both i, we loop back to either Negotiation or TreatResidual, depending whether $\mathbf{S}_i$ are trim or not.

Finally, observe that the condition that checks whether $\perp$ is a state in some $\mathbf{S}_i$ will never be true if $\Sigma^{\mathrm{s}}_{\mathrm{uc}}$ is empty – because then, by the definition of PCSynth, the automata $\mathbf{S}_i$ would be trim. In this case, the scheme in Fig. 9 would be reduced

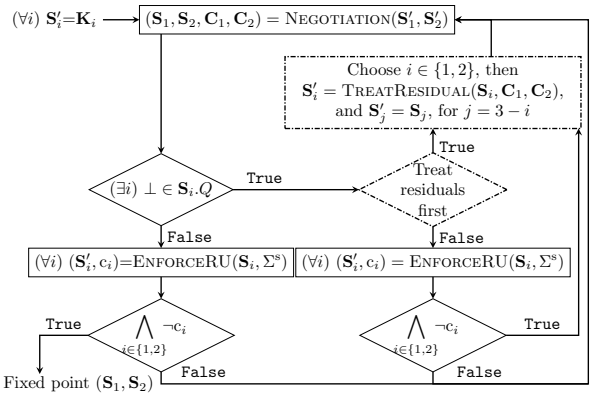

Fig. 9. Scheme for the synthesis of the automata $\mathbf{S}_i$.

to a loop with only Negotiation and EnforceRU– see [16] – as we would not have the decisions shown as dashed blocks in the figure. However, though the emptiness of $\Sigma^{\mathrm{s}}_{\mathrm{uc}}$ is a sufficient condition for this to happen, it is not necessary.

A. Exploratory Procedure for Synthesis

From the foregoing discussion, we see in Fig. 9 that, in general, we are left with two decisions. The first is to select one of the auxiliary functions to be called after negotiation – if some of its outputs is not trim – and the second is about where to prevent $\perp$ to be reached when we call TreatResidual, as discussed in Sec. IV-D.

The dilemma, mentioned in the beginning of this section, is that we cannot know beforehand which choice leads to the more permissive solution – although all the options lead to correct results, as we will prove later in this section. So it is not obvious how to extract a rule for each decision, as it depends on each particular plant model. Even if we could compute the resulting pair of automata $\mathbf{S}_i$ for all possible combinations of choices, it would not be possible to *guarantee* that one of them enforces the language $\sup\mathcal{C}(K_1 \parallel K_2)$. In fact, different sequences of choices may lead to incomparable solutions; that is, we may not even be able to compare these pairs in terms of maximal permissiveness.

Therefore, what we propose here is an exploratory procedure, based on the scheme in Fig. 9, that tries the combinations of decisions until we find nonempty automata that can realise, via Def. 5, the desired local supervisors. This procedure, given in Proc. 4, is denoted by CBSS, as an initialism for *Contract-Based Supervisor Synthesis*.

We cannot guarantee maximal permissiveness, but we also do not wish for trivial solutions for our synthesis problem; besides, we do not want to explore all possibilities mentioned so far, because even if we do so, we might not be able to compare these solutions. Thus, as it is often done in Supervisory Control Theory – e.g., in [17] – it is natural to have a specification of a minimal behaviour, which can be given as a trim automaton $\mathbf{L}_i$. Just as each plantified specification $\mathbf{K}_i$ is an upper bound for the controlled behaviour of the corresponding subsystem, in the sense that $S_i \subseteq K_i$ should be satisfied, this minimal behaviour would be the local lower bound, meaning $L_i \subseteq S_i$ should hold. We then use this information on our procedure, by ceasing to explore a combination of decisions that lead to

Procedure 4 CBSS

Require: Automata $\mathbf{S}_1^0$ and $\mathbf{S}_2^0$. **Optional:** Trim automata $\mathbf{L}_1$ and $\mathbf{L}_2$ (otherwise they are such that $L_1 = L_2 = \{\epsilon\}$).

1: $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{C}_1, \mathbf{C}_2) \leftarrow$ NEGOTIATION$(\mathbf{S}_1^0, \mathbf{S}_2^0)$
2: $v \leftarrow$ (CHECKLB$(\mathbf{S}_1)$ and CHECKLB$(\mathbf{S}_2)$)
3: **if** $\neg v$ **then return** $(\mathbf{S}_1, \mathbf{S}_2, v)$
4: $b \leftarrow$ (TEST$\perp(\mathbf{S}_1)$ or TEST$\perp(\mathbf{S}_2)$)
5: **if** b **then** $t \leftarrow$ DECIDE() **else** $t \leftarrow$ `False`
6: **if** t **then**
7: $\quad(\hat{\mathbf{S}}_1, \hat{\mathbf{S}}_2, v) \leftarrow$ NEXTTREATRESIDUAL$(\mathbf{S}_1, \mathbf{S}_2, \mathbf{C}_1, \mathbf{C}_2)$
8: $\quad$**if** $\neg v$ **then**
9: $\quad\quad(\hat{\mathbf{S}}_1, \hat{\mathbf{S}}_2, v, c) \leftarrow$ NEXTENFORCERU$(\mathbf{S}_1, \mathbf{S}_2)$
10: $\quad\quad$**if** $\neg c$ **then** $v \leftarrow$ `False`
11: **else**
12: $\quad(\hat{\mathbf{S}}_1, \hat{\mathbf{S}}_2, v, c) \leftarrow$ NEXTENFORCERU$(\mathbf{S}_1, \mathbf{S}_2)$
13: $\quad$**if** b and (v xor c) **then**
14: $\quad\quad(\hat{\mathbf{S}}_1, \hat{\mathbf{S}}_2, v) \leftarrow$ NEXTTREATRESIDUAL$(\mathbf{S}_1, \mathbf{S}_2, \mathbf{C}_1, \mathbf{C}_2)$
15: **return** $(\hat{\mathbf{S}}_1, \hat{\mathbf{S}}_2, v)$

16: **procedure** NEXTTREATRESIDUAL$(\mathbf{S}_1^0, \mathbf{S}_2^0, \mathbf{C}_1, \mathbf{C}_2)$
17: $\quad\mathbf{S}_1 \leftarrow$ TREATRESIDUAL$(\mathbf{S}_1^0, \mathbf{C}_1, \mathbf{C}_2)$
18: $\quad(\mathbf{S}_1, \mathbf{S}_2, v) \leftarrow$ NEWBRANCH$(\mathbf{S}_1, \mathbf{S}_2^0)$
19: $\quad$**if** $\neg v$ **then**
20: $\quad\quad\mathbf{S}_2 \leftarrow$ TREATRESIDUAL$(\mathbf{S}_2^0, \mathbf{C}_1, \mathbf{C}_2)$
21: $\quad\quad(\mathbf{S}_1, \mathbf{S}_2, v) \leftarrow$ NEWBRANCH$(\mathbf{S}_1^0, \mathbf{S}_2)$
22: $\quad$**return** $(\mathbf{S}_1, \mathbf{S}_2, v)$

23: **procedure** NEXTENFORCERU$(\mathbf{S}_1^0, \mathbf{S}_2^0)$
24: $\quad(\mathbf{S}_1, c_1) \leftarrow$ ENFORCERU$(\mathbf{S}_1^0, \Sigma^{\mathrm{s}})$
25: $\quad(\mathbf{S}_2, c_2) \leftarrow$ ENFORCERU$(\mathbf{S}_2^0, \Sigma^{\mathrm{s}})$
26: $\quad v \leftarrow$ `True` and $c \leftarrow (c_1$ or $c_2)$
27: $\quad$**if** c **then** $(\mathbf{S}_1, \mathbf{S}_2, v) \leftarrow$ NEWBRANCH$(\mathbf{S}_1, \mathbf{S}_2)$
28: $\quad$**return** $(\mathbf{S}_1, \mathbf{S}_2, v, c)$

29: **procedure** NEWBRANCH$(\mathbf{S}_1^0, \mathbf{S}_2^0)$
30: $\quad\mathbf{S}_1 \leftarrow \mathbf{S}_1^0$ and $\mathbf{S}_2 \leftarrow \mathbf{S}_2^0$
31: $\quad v \leftarrow$ (CHECKLB$(\mathbf{S}_1)$ and CHECKLB$(\mathbf{S}_2)$)
32: $\quad$**if** v **then** $(\mathbf{S}_1, \mathbf{S}_2, v) \leftarrow$ CBSS$(\mathbf{S}_1, \mathbf{S}_2)$
33: $\quad$**return** $(\mathbf{S}_1, \mathbf{S}_2, v)$

the lower bound specification to be disrespected, and trying the next combination. In Proc. 4, this is done by the function CHECKLB, which returns true iff $L_i \subseteq S_i$; the boolean v that stores this information stands for *valid*.

The execution of CBSS is illustrated by the tree in Fig. 10. This multiple recursive procedure explores the options in a depth-first-search manner. The particularity here is that we cannot know a priori the depth of each branch; besides, this depth is variable from branch to branch, i.e., from the root to a leaf in the tree, the depth depends on previous decisions taken to reach that particular leaf. These leaves represent nodes where the search is interrupted either due to a violation of the specification $\mathbf{L}_i$ – i.e., when v is false – or when the procedure finally finds a pair of trim automata $\mathbf{S}_i$ as desired, that is, compatible with equivalent contracts, satisfying relative unambiguity and respecting $\mathbf{L}_i$.

Initially, as in Fig. 9, CBSS calls NEGOTIATION, and then checks whether its outputs are trim. This check is done by TEST$\perp$, which returns true iff $\perp$ is a state in its inputs; b (line 4) stands for *blocking*. If b is true, the function DECIDE chooses – randomly or using any kind of heuristics – whether or not to prioritise TREATRESIDUAL over ENFORCERU– that is, it decides which function to call next; t stands for *trimming* $\perp$. If b is false, as discussed before, there is no choice and ENFORCERU, not TREATRESIDUAL, should be

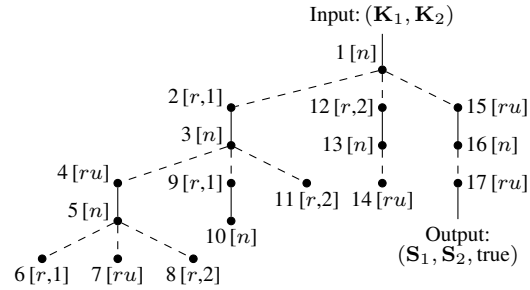

Fig. 10. Tree illustrating the search by CBSS, from Proc. 4. The number next to each node represents the order in which the procedure explores different combinations of decisions. The letters $\boldsymbol{n}$, $\boldsymbol{r}$, and $\boldsymbol{ru}$ denote, respectively, the execution of the functions NEGOTIATION, TREATRESIDUAL, and ENFORCERU.

called. Based on this choice, either NEXTTREATRESIDUAL or NEXTENFORCERU call NEWBRANCH, which then starts a new recursion. When a leaf – as described in the preceding paragraph – is reached, the search on that branch stops. The outputs of both NEWBRANCH and CBSS are a pair of automata $\mathbf{S}_i$ and the boolean v. A call of one of these functions return v as true if and only if $\mathbf{S}_i$ are as desired; in this case, v and $\mathbf{S}_i$ are returned through all the recursions till the first call of CBSS. If the returned v is false and there are still new combinations of decisions to be searched, a new branch – i.e., a new recursion – is started; otherwise, the recursion ends without the desired automata being found, and the first call of CBSS returns v as false.

Finally, by combining multiple previous results, we have the following soundness result of Proc. 4. It is the main implication of this section, for it shows that, if found, the fully local supervisors impose a globally nonblocking closed-loop behaviour that respects given local specifications.

Theorem 2: Consider a plant composed of two subsystems $\mathbf{M}_i$ with plantified specifications $\mathbf{K}_i$, as in Sec. III-A. Let $(\mathbf{S}_1, \mathbf{S}_2, v) = \mathrm{CBSS}(\mathbf{K}_1, \mathbf{K}_2)$. If v is true, let f_i be the local supervisors defined from $\mathbf{S}_i$ via Def. 5. Then, it holds that

1) $\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2) \in \mathcal{C}(K_1 \parallel K_2)$, and
2) $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2)}$,

where controllability is taken with respect to $\mathcal{L}(\mathbf{M}_1 \parallel \mathbf{M}_2)$ and $\Sigma_{1,\mathrm{uc}}^{\mathrm{p}} \cup \Sigma_{2,\mathrm{uc}}^{\mathrm{p}} \cup \Sigma_{\mathrm{uc}}^{\mathrm{s}}$. □

Although we may need to sacrifice maximal permissiveness in order to respect privacy, our last theorem shows that in certain cases we can guarantee this trait is still preserved.

Theorem 3: Let $\mathbf{M}_i$ and $\mathbf{K}_i$ be subsystems and their plantified specifications, as in Sec. III-A, and $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{C}_1, \mathbf{C}_2) =$ NEGOTIATION$(\mathbf{K}_1, \mathbf{K}_2)$. If $\mathbf{S}_i$ are trim automata, with S_i unambiguous as stated in Def. 6, and if f_i is defined from $\mathbf{S}_i$ via Def. 5, we have that

1) $\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2) = \sup\mathcal{C}(K_1 \parallel K_2)$, and
2) $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2)}$,

where controllability is taken as in Thm. 2. □

Remark 9: It is unclear whether NEGOTIATION from Proc. 1 and, consequently, CBSS from Proc. 4 always terminate. The combination of PCSYNTH with the extraction and exchange of contracts $\mathbf{C}_i$ during NEGOTIATION may cause an increase of the size of the automata $\mathbf{S}_i$ from one round to the next – where each round starts at line 5 in Proc. 1. We then cannot use as argument for the termination of these procedures

the monotonicity of the size of the automata they manipulate. While this leaves a (non)termination proof for future work, we remark that we were not able to construct a counterexample for the termination of NEGOTIATION and CBSS. Moreover, as discussed in the next section, both these procedures terminated on all the performed experiments. Finally, we remark that the complexity on each round of NEGOTIATION and CBSS is dominated by the computation of the observer automata, which in the worst case is exponential in the size of the automata $\mathbf{S}_i$. These observers are required in Proc. 1 for the contract extraction and in Proc. 3, used by Proc. 4, for the check and possible enforcement of relative unambiguity. □

VII. EXPERIMENTAL RESULTS

We demonstrate the suitability of our framework by evaluating our approach over randomly generated plants. We implemented Proc. 4 for CBSS – including Proc. 1 for NEGOTIATION and all the other procedures here presented – in the tool `Supremica` [20]. For the generation of random examples, we used the tool `MDESops` [21].

We present in Table I the results of running CBSS over 8 different cases corresponding to the columns of the table (except for the first). For each case, we randomly generated 100 examples, i.e., 100 plants composed of 2 subsystems each. The number of states in the table is given per subsystem; thus, for example, cases with 1000 states per subsystem correspond to plants that could reach a million states. Row (1) shows the *average* runtime for CBSS taken from 100 examples. Row (2) shows the percentage of examples for which CBSS returned nonempty solutions. Rows (3) to (8) show the *average* of the following data taken among the examples that returned nonempty solutions: (3) runtime for CBSS; (4) and (5) average number of states per supervisor $\mathbf{S}_i$ and contract $\mathbf{C}_i$ outputted by CBSS, respectively; (6) number of NEGOTIATION calls during CBSS; (7) average runtime per NEGOTIATION call; (8) average number of rounds exchanging contracts per NEGOTIATION call. Runtimes are shown in milliseconds (ms).

For all the examples considered in this experiment, the supervisors outputted by the first call of NEGOTIATION did not satisfy relative unambiguity. When we increase the number of shared events between the subsystems, this property becomes stronger, and because it is not immediately satisfied after NEGOTIATION, it needs to be enforced, which leads to a higher number of empty solutions. Finally, we remark that our procedure terminated on all our experiments.

	Number of States per Subsystem							
	50		200		1000			
	Number of shared events / Size of each local alphabet Σ_i							
	3/15	5/15	3/15	5/15	4/20	5/20	6/20	7/20
(1)	69	84	999	1132	45202	86545	50043	86955
(2)	91	72	66	33	78	58	38	12
(3)	71	86	1098	1515	47347	76271	59395	65708
(4)	47	51	186	167	922	900	845	789
(5)	1	1	1	1	1	1	1	1
(6)	2	2	3	3	3	3	3	3
(7)	25	24	234	252	9008	8356	9408	9931
(8)	2	2	2	2	2	2	2	2

TABLE I
DATA OBTAINED FROM RUNNING CBSS FOR 100 RANDOMLY GENERATED EXAMPLES PER CASE (COLUMNS).

VIII. CONCLUSION

In this paper, we address the problem of finding local decentralised supervisors for a distributed plant composed of two subsystems that synchronise over shared events, while preserving privacy with respect to events that are local but nonshared, regarded as private. We consider the controllability of the shared events may not be the same in both subsystems, and that they have their own local specifications to be satisfied.

The distinctive aspect of our method, if compared to other distributed and decentralised approaches, is privacy, as we consider the problem of partial observation *not only* during the execution of the supervisors, but also during their synthesis. Privacy is respected for the following reasons. During synthesis, no information about private events is shared between the subsystems: the contracts negotiated between them are represented solely in terms of the events they share, and further refinements applied in the supervisors are done fully locally. As a consequence, we highlight that, apart from what is expressed in the contracts, the complete model of each subsystem remains unknown to the other subsystem. Moreover, once the desired supervisors are found, they can be executed in parallel without the need of a coordinator to solve conflicts, nor the communication of private events – which differs from the use of signalling bits proposed here for the case where the supervisors have different controllability settings over shared events, as discussed in Rem. 6.

Our approach is sound, as we prove that the composition of the subsystems in closed-loop with their local supervisors enforces a nonblocking behaviour that satisfies the local specifications (Thm. 2). It is not complete, since the problem of synthesising local, decentralised supervisors is known to be undecidable [22], [23]. This is due to partial observation: even though, in our setting, a supervisor can observe all its local events, it cannot observe private events from the other subsystem it synchronises with. In particular, maximal permissiveness can not always be achieved. Yet for the cases our approach returns nonempty solutions, we can identify when they are provenly maximally permissive (Thm. 3).

We restrict our attention to systems composed of only two processes. Although this imposes a limitation to our framework, it has been applied in [24] for the synthesis of two stealthy sensor and actuator attackers that cooperate in achieving their goals, while requiring some privacy in terms of their local behaviour. Moreover, we are currently working on expanding our approach for multiple processes. This is not a trivial task, as the way the processes are connected among themselves in terms of the events they share impacts: (i) how they should be composed into groups to negotiate contracts; (ii) over which observers the local property for nonconflict, namely relatively unambiguity, should be checked and enforced; (iii) whether or not groups require a somewhat-local coordinator among themselves to prevent conflict, on top of the negotiated contracts and local properties; and (iv)

privacy, as a consequence of all the aspects listed above.

Appendix I
Proofs

For the proofs, and for an automaton $\mathbf{\Lambda}$ over Σ, we adopt the notation $\sup\mathcal{C}_{\mathbf{\Lambda},\Sigma_{\mathrm{u}}}$ to indicate that controllability is taken with respect to $\mathcal{L}(\mathbf{\Lambda})$, and to an uncontrollable alphabet $\Sigma_{\mathrm{u}} \subseteq \Sigma$.

Supplementary Remark 1: For automata $\mathbf{M}$ and $\mathbf{\Lambda}$ over Σ such that $\Lambda \subseteq \mathcal{L}(\mathbf{M})$, and for $\Sigma'_{\mathrm{uc}} \subseteq \Sigma_{\mathrm{uc}} \subseteq \Sigma$, we have that $\sup\mathcal{C}_{\mathbf{M},\Sigma_{\mathrm{uc}}}(\Lambda) \subseteq \sup\mathcal{C}_{\mathbf{M},\Sigma'_{\mathrm{uc}}}(\Lambda)$.

Remark 2. If $\mathbf{K}_i$ is a plantified specification for the subsystem $\mathbf{M}_i$, and $\mathbf{S}_i = \textsc{PCSynth}(\mathbf{K}_i)$, then we have that $S_i = \sup\mathcal{C}(K_i)$ with respect to $\mathcal{L}(\mathbf{M}_i)$ and $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}}$. Moreover, if we take controllability with respect to $\mathcal{L}(\mathbf{M}_i)$ and $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$, then $\sup\mathcal{C}(S_i) = \sup\mathcal{C}(K_i)$.

Rem. 2 follows from the definition of PCSynth, and from Rem. 1 and Sup. Rem. 1.

Supplementary Remark 2: Let $\mathbf{S}^0_i$ be any pair of automata that satisfies requirements (1) and (2) from Cond. 1. Then $\sup\mathcal{C}_{\mathbf{M}_i,\Sigma^{\mathrm{p}}_{i,\mathrm{uc}}}(S^0_i) = \sup\mathcal{C}_{\mathbf{S}^0_i,\Sigma^{\mathrm{p}}_{i,\mathrm{uc}}}(S^0_i)$.

Supplementary Remark 3: Let $\mathbf{S}^0_i$ be any pair of automata that satisfies Cond. 1. Then, for $\mathbf{S}_i = \textsc{PCSynth}(\mathbf{S}^0_i)$, we have that the pair $(\mathbf{S}_1, \mathbf{S}_2)$ also satisfies Cond. 1.

Its is straightforward to prove Sup. Rem. 3 by considering Cond. 1, the definition of PCSynth, and Sup. Rem. 2.

Supplementary Remark 4: Let $\mathbf{S}^0_i$ be any pair of automata that satisfies Cond. 1. Then, for $\mathbf{S}_i = \mathbf{S}^0_i \parallel \mathbf{O}_{\mathbf{S}^0_j,\Sigma^{\mathrm{s}}}$ and $i, j \in \{1, 2\}$, with $i \neq j$, we have that $(\mathbf{S}_1, \mathbf{S}_2)$ also satisfies Cond. 1.

Supplementary Remark 5: Suppose that a pair of automata $\mathbf{S}^0_i$ satisfying Cond. 1 is given as input for Negotiation. Denote by $\mathbf{S}'_{i,k}$ the automaton $\mathbf{S}'_i$ in Proc. 1 at the beginning of each round $k \geq 1$, which we consider to be at line 5. Then, we have that $(\mathbf{S}'_{1,k}, \mathbf{S}'_{2,k})$ also satisfies Cond. 1 for all $k \geq 1$.

From Sup. Rem. 3 and 4, it is straightforward to show by induction Sup. Rem. 5.

Lemma 1. Let $\mathbf{S}^0_i$ be any pair of automata that satisfies Cond. 1, and $\mathbf{S}_i$ be the outputs of Proc. 1 with $\mathbf{S}^0_i$ as its inputs, namely, $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{C}_1, \mathbf{C}_2) = \textsc{Negotiation}(\mathbf{S}^0_1, \mathbf{S}^0_2)$. Then the pair $\mathbf{S}_i$ also satisfies Cond. 1, and both automata are compliant with the contracts $\mathbf{C}_i$, which are equivalent.

Proof:

The fixed point of Negotiation is reached when $cond =$ `False`, which occurs when $\mathbf{S}_i = \mathbf{S}_i \parallel \mathbf{O}_{\mathbf{S}_j,\Sigma^{\mathrm{s}}}$ for all i, with $j \neq i$. By Sup. Rem. 5, it then follows that $(\mathbf{S}_1, \mathbf{S}_2)$ satisfies Cond. 1. Moreover, we also have that $P_{i\mathrm{s}}(S_i) \subseteq P_{j\mathrm{s}}(S_j)$. Thus, $P_{1\mathrm{s}}(S_1) = P_{2\mathrm{s}}(S_2)$; analogously, $P_{1\mathrm{s}}(\mathcal{L}(\mathbf{S}_1)) = P_{2\mathrm{s}}(\mathcal{L}(\mathbf{S}_2))$. Hence, $\mathbf{C}_1$ is equivalent to $\mathbf{C}_2$, and $\mathbf{S}_i$ are compliant with these contracts. ∎

Supplementary Remark 6: Given languages A, B and $C = \overline{C}$ over $\Sigma = \Sigma_{\mathrm{c}} \dot\cup \Sigma_{\mathrm{uc}}$ such that $A, B \subseteq C$, we have that

$$\sup\mathcal{C}(A) \subseteq B \;\Leftrightarrow\; \sup\mathcal{C}(A) \subseteq \sup\mathcal{C}(B)\,,$$

where controllability is taken with respect to Σ_{uc} and C.

Supplementary Lemma 1: In the context of the plant defined in Sec. III-A, and for any languages Λ_i over Σ_i such that $\Lambda_i \subseteq K_i$, $i \in \{1, 2\}$, we have that

$$P_i\big(\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)\big) \subseteq \sup\mathcal{C}_{\mathbf{M}_i,\Sigma^{\mathrm{p}}_{i,\mathrm{uc}}}(\Lambda_i),$$

where we denote by P_i the projection $P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma^*_i$, the global plant by $\mathbf{M} = \mathbf{M}_1 \parallel \mathbf{M}_2$, and by Σ^p_{uc} the alphabet $\Sigma^{\mathrm{p}}_{1,\mathrm{uc}} \dot\cup \Sigma^{\mathrm{p}}_{2,\mathrm{uc}}$.

Proof:

(i) $\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2) \subseteq \Lambda_1 \parallel \Lambda_2$, so $\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2) \subseteq P_i^{-1}(\Lambda_i) \Rightarrow P_i\big(\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)\big) \subseteq \Lambda_i$.

(ii) By the definition of $\sup\mathcal{C}$ and controllability, we have that, for all $u \in \overline{\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)}$ and $\sigma \in \Sigma^p_{\mathrm{uc}}$, if $u\sigma \in \mathcal{L}(\mathbf{M})$, then $u\sigma \in \overline{\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)}$. Besides, $P_i(u) \in P_i\big(\overline{\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)}\big) = P_i\big(\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)\big)$, and $u \in \mathcal{L}(\mathbf{M})$, because $\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2) \subseteq K_1 \parallel K_2 \subseteq \mathcal{L}(\mathbf{M})$, which is prefix-closed by definition. Note that $\sigma \in \Sigma^{\mathrm{p}}_{i,\mathrm{uc}}$ for some i, that is, σ is a local event and hence $\sigma \notin \Sigma_j$ for $j \neq i$; then $P_i(u)\sigma \in \mathcal{L}(\mathbf{M}_i) \Leftrightarrow u\sigma \in \mathcal{L}(\mathbf{M})$. Now, by the definition of $\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)$, we have that $P_i(u)\sigma \in \mathcal{L}(\mathbf{M}_i) \Leftrightarrow \underline{u\sigma \in \mathcal{L}(\mathbf{M}) \Rightarrow u\sigma \in \overline{\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)}} \Rightarrow P_i(u)\sigma \in \overline{P_i\big(\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)\big)}$, implying $P_i\big(\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)\big)$ is controllable with respect to $\mathcal{L}(\mathbf{M}_i)$ and $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}}$.

From (i) and (ii), we have that $P_i\big(\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(\Lambda_1 \parallel \Lambda_2)\big) \subseteq \sup\mathcal{C}_{\mathbf{M}_i,\Sigma^{\mathrm{p}}_{i,\mathrm{uc}}}(\Lambda_i)$. ∎

Theorem 1. In the context of Lem. 1, we have that $\sup\mathcal{C}(S_1 \parallel S_2) = \sup\mathcal{C}(S^0_1 \parallel S^0_2)$, where $\sup\mathcal{C}$ is taken with respect to $\mathcal{L}(\mathbf{M}_1 \parallel \mathbf{M}_2)$ and $\Sigma^{\mathrm{p}}_{1,\mathrm{uc}} \cup \Sigma^{\mathrm{p}}_{2,\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$.

Proof:

($\Rightarrow$) From Sup. Rem. 5, we have that $S_1 \parallel S_2 \subseteq S^0_1 \parallel S^0_2$; thus, $\sup\mathcal{C}(S_1 \parallel S_2) \subseteq \sup\mathcal{C}(S^0_1 \parallel S^0_2)$.

($\Leftarrow$) Let us denote $\Sigma^{\mathrm{p}}_{1,\mathrm{uc}} \dot\cup \Sigma^{\mathrm{p}}_{2,\mathrm{uc}}$ by Σ^p_{uc}. From Sup. Rem. 5, we have that $(\mathbf{S}'_{1,k}, \mathbf{S}'_{2,k})$ satisfies Cond. 1, for all $k \geq 1$. As $\mathbf{S}^0_i$ are the inputs for Negotiation, define $\mathbf{S}'_{i,0} = \mathbf{S}^0_i$; thus, $(\mathbf{S}'_{1,0}, \mathbf{S}'_{2,0})$ satisfies Cond. 1 as well. We can, then, apply Sup. Rem. 2, together with the definition of PCSynth, to write:

$$\begin{aligned}
&S'_{1,k} \parallel S'_{2,k} \\
&\quad = \sup\mathcal{C}_{\mathbf{M}_1,\Sigma^{\mathrm{p}}_{1,\mathrm{uc}}}(S'_{1,k-1}) \parallel P_{2\mathrm{s}}(\sup\mathcal{C}_{\mathbf{M}_2,\Sigma^{\mathrm{p}}_{2,\mathrm{uc}}}(S'_{2,k-1})) \parallel \\
&\qquad \sup\mathcal{C}_{\mathbf{M}_2,\Sigma^{\mathrm{p}}_{2,\mathrm{uc}}}(S'_{2,k-1}) \parallel P_{1\mathrm{s}}(\sup\mathcal{C}_{\mathbf{M}_1,\Sigma^{\mathrm{p}}_{1,\mathrm{uc}}}(S'_{1,k-1})) \\
&\quad = \sup\mathcal{C}_{\mathbf{M}_1,\Sigma^{\mathrm{p}}_{1,\mathrm{uc}}}(S'_{1,k-1}) \parallel \sup\mathcal{C}_{\mathbf{M}_2,\Sigma^{\mathrm{p}}_{2,\mathrm{uc}}}(S'_{2,k-1}) \\
&\quad \supseteq \sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}}}(S'_{1,k-1} \parallel S'_{2,k-1})\,. \quad \text{(by Sup. Lem. 1)} \\
&\quad \supseteq \sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}}(S'_{1,k-1} \parallel S'_{2,k-1})\,. \quad \text{(by Sup. Rem. 1).}
\end{aligned}$$

Then, from Sup. Rem. 6, we have that $\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}}(S'_{1,k-1} \parallel S'_{2,k-1}) \subseteq \sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}}(S'_{1,k} \parallel S'_{2,k})$. Since this is valid for all $k \geq 1$, if follows that $\sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}}(S^0_1 \parallel S^0_2) \subseteq \sup\mathcal{C}_{\mathbf{M},\Sigma^p_{\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}}(S'_{1,\tilde{k}} \parallel S'_{2,k})$. Finally, as the fixed point $S_i = S'_{1,\tilde{k}}$ for some $\tilde{k} \geq 1$, we conclude that $\sup\mathcal{C}(S^0_1 \parallel S^0_2) \subseteq \sup\mathcal{C}(S_1 \parallel S_2)$. ∎

Corollary 1. Given the premisses of Lem. 1, if $\mathbf{S}_i$ are trim automata and S_1 and S_2 are nonconflicting, we have that $S_1 \parallel S_2 = \sup\mathcal{C}(S^0_1 \parallel S^0_2)$, where $\sup\mathcal{C}$ is taken with respect to $\mathcal{L}(\mathbf{M}_1 \parallel \mathbf{M}_2)$ and $\Sigma^{\mathrm{p}}_{1,\mathrm{uc}} \cup \Sigma^{\mathrm{p}}_{2,\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$.

Proof:

Each language S_i is controllable with respect to $\mathcal{L}(\mathbf{M}_i)$ and $\Sigma^{\mathrm{p}}_{i,\mathrm{uc}}$, because $\mathbf{S}_i$ is the output of the last call of the PCSynth function during Negotiation, and by Sup. Rem. 2 and 5. We assume automata $\mathbf{S}_i$ to be nonblocking, so, by PCSynth definition, and since the blocking state $\perp$ is not present, we have that S_i is also controllable with respect to $\Sigma^{\mathrm{s}}_{\mathrm{uc}}$. Now, let

us denote by S the composition $S_1 \parallel S_2$. The languages S_1 and S_2 are nonconflicting, so $\overline{S} = \overline{S_1} \parallel \overline{S_2}$. The synchronisation in $\overline{S_1} \parallel \overline{S_2}$ only occurs over shared events, which are either in $\Sigma^{\mathrm{s}}_{\mathrm{uc}}$ or are considered globally controllable; thus, it is straightforward to show that S is controllable with respect to $\mathcal{L}(\mathbf{M}_1 \parallel \mathbf{M}_2)$ and $\Sigma^{\mathrm{p}}_{1,\mathrm{uc}} \cup \Sigma^{\mathrm{p}}_{2,\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$. Then, for there is no language L such that $S \subset L \subseteq S$, we have that $S = \sup\mathcal{C}(S)$. Finally, by Thm. 1, it follows that $S = \sup\mathcal{C}(S_1^0 \parallel S_2^0)$. ■

Lemma 2. In the context of Def. 5, we have that

1) $\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2) = S_1 \parallel S_2$ and
2) $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{S_1} \parallel \overline{S_2}$.

Proof:

Denote $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2)$ by L, and $\Sigma_1 \cup \Sigma_2$ by Σ.

(a) $\overline{S_1} \parallel \overline{S_2} \subseteq \mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2)$:

If there exists $i \in \{1,2\}$ such that $S_i = \emptyset$, then $\overline{S_1} \parallel \overline{S_2} = \emptyset$ and it trivially holds. Consider then $\overline{S_i} \neq \emptyset$ for all $i \in \{1,2\}$. Then, $\epsilon \in \overline{S_1} \parallel \overline{S_2}$, and $\epsilon \in L$ by definition of $\mathcal{L}(f_i/\mathbf{M}_i)$.

Let us say $\overline{S_1} \parallel \overline{S_2} \supset \{\epsilon\}$ and prove that $s \in L$ for any $s \in \overline{S_1} \parallel \overline{S_2}$. This proof is by induction over the length of s. Note that $\overline{S_i} \subseteq \mathcal{L}(\mathbf{M}_i)$, so $s \in \mathcal{L}(\mathbf{M}_1) \parallel \mathcal{L}(\mathbf{M}_2)$, i.e., $P_i(s) \in \mathcal{L}(\mathbf{M}_i)$ for all i.

For length 0 ($s = \epsilon$), it holds. Assume it holds for some $s = w > \epsilon$, and denote $P_i(w) \in \overline{S_i}$ by w_i for all i. Then, for any $\sigma \in \Sigma$, take $s = w\sigma$. If $\sigma \in \Sigma^{\mathrm{p}}_i$ for some i, then $w_i\sigma \in \overline{S_i}$ and $\sigma \notin \Sigma^{\mathrm{s}}_{j,\mathrm{cp}}$, so by definition of f_i, $w_i\sigma \in \mathcal{L}(f_i/\mathbf{M}_i)$; moreover, $P_j(\sigma) = \epsilon$; thus, $w\sigma \in L$. If $\sigma \in \Sigma^{\mathrm{s}} \backslash (\Sigma^{\mathrm{s}}_{1,\mathrm{cp}} \cup \Sigma^{\mathrm{s}}_{2,\mathrm{cp}})$, then by definition of f_i and for all i, we have that $w_i\sigma \in \mathcal{L}(f_i/\mathbf{M}_i)$; hence, $s \in L$. If $\sigma \in \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$ for some i, then $\mathrm{d}^\sigma \notin d_i(w_i)$, so $\sigma \in f_j\big(w_j, d_i(w_i)\big)$; thus, $s \in L$.

(b) $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) \subseteq \overline{S_1} \parallel \overline{S_2}$:

By definition of $\mathcal{L}(f_i/\mathbf{M}_i)$ and f_i, we have that $L \neq \emptyset$. Thus, $\epsilon \in L$, and we know $\epsilon \in \overline{S_1} \parallel \overline{S_2}$. Let us say $\overline{S_1} \parallel \overline{S_2} \supset \{\epsilon\}$ and, for any $s \in L$, prove by induction over the length of s that $s \in \overline{S_1} \parallel \overline{S_2}$.

For length 0 ($s = \epsilon$), it holds. Assume it holds for some $s = w > \epsilon$, and denote $P_i(w) \in \mathcal{L}(f_i/\mathbf{M}_i)$ by w_i for all i. Then, for any $\sigma \in \Sigma$, take $s = w\sigma$. If $\sigma \in \Sigma^{\mathrm{p}}_i$ for some i, then $w_i\sigma \in \overline{S_i}$ by definition of f_i, and since $P_j(\sigma) = \epsilon$, we have that $s \in \overline{S_1} \parallel \overline{S_2}$. If $\sigma \in \Sigma^{\mathrm{s}} \backslash (\Sigma^{\mathrm{s}}_{1,\mathrm{cp}} \cup \Sigma^{\mathrm{s}}_{2,\mathrm{cp}})$, then by definition of f_i and for all i, we have that $w_i\sigma \in \overline{S_i}$; hence, $s \in \overline{S_1} \parallel \overline{S_2}$. If $\sigma \in \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$ for some i: suppose $w_i\sigma \notin \overline{S_i}$; then $\mathrm{d}^\sigma \in d_i(w_i)$, so $\sigma \notin f_j\big(w_j, d_i(w_i)\big) \Rightarrow w_j\sigma \notin \mathcal{L}(f_j/\mathbf{M}_j) \Rightarrow w\sigma \notin L$, which is a contraction; thus, $w_i\sigma \in \overline{S_i}$ and $\mathrm{d}^\sigma \notin d_i(w_i) \Rightarrow w_j\sigma \in \overline{S_j}$. Hence, $s \in \overline{S_1} \parallel \overline{S_2}$.

(c) $S_1 \parallel S_2 \subseteq \mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2)$:

This result follows immediately from (a) and from the definition of $\mathcal{L}_m(f_i/\mathbf{M}_i)$.

(d) $\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2) \subseteq S_1 \parallel S_2$:

This result follows immediately from (b) and from the fact that S_i are relatively closed with respect to $\mathcal{L}_m(\mathbf{M}_i)$. ■

Corollary 2. The global behaviour of the plant in closed loop with the supervisors f_i from Def. 5 is nonblocking, that is, $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2)}$, if and only if S_1 and S_2 are nonconflicting.

Proof:

It follows from the definition of conflict and Lem. 2. ■

Proposition 1. For automata $\mathbf{S}_1$ and $\mathbf{S}_2$ as in Lem. 1, if they are trim, and if their marked languages S_1 and S_2 are unambiguous with respect to Σ^{s} and the natural projection $P_{i\mathrm{s}} : \Sigma_i^* \to \Sigma^{\mathrm{s}*}$, then these languages are nonconflicting.

Proof:

For any language S_i, it is always true (and trivial to show) that $\overline{S_1 \parallel S_2} \subseteq \overline{S_1} \parallel \overline{S_2}$. Let us then prove the other direction, i.e., that $\overline{S_1} \parallel \overline{S_2} \subseteq \overline{S_1 \parallel S_2}$. Once more, we use the definitions of P_i and P_{s} as in the proof of Lemma 2.

Let t be any string in $\overline{S_1} \parallel \overline{S_2} = P_1^{-1}(\overline{S_1}) \cap P_2^{-1}(\overline{S_2})$. Then, for any $i \in \{1,2\}$ there exists $u_i \in \Sigma_i^*$ such that $t_iu_i \in S_i$, where $t_i = P_i(t)$. If $P_{1\mathrm{s}}(u_1) = P_{2\mathrm{s}}(u_2) = \epsilon$, take $w \in \Sigma^*$ such that $w = u_1u_2$. Then, $P_i(tw) = t_iu_i \in S_i$, so $tw \in S_1 \parallel S_2$ and hence $t \in \overline{S_1 \parallel S_2}$.

Else, there is i such that $P_{i\mathrm{s}}(u_i) \neq \epsilon$. Now, because $P_{1\mathrm{s}}(S_1) = P_{2\mathrm{s}}(S_2)$, there exist $\hat{t}_j$ and $\hat{u}_j$ such that $P_{j\mathrm{s}}(\hat{t}_j) = P_{i\mathrm{s}}(t_i)$, $P_{j\mathrm{s}}(\hat{u}_j) = P_{i\mathrm{s}}(u_i)$, and $\hat{t}_j\hat{u}_j \in S_j$ (note that we might have $\hat{t}_j \neq t_j = P_j(t)$). Thus, in summary, we have that $t_j \in \overline{S_j}$ (from the initial assumption), $\hat{t}_j \in \overline{S_j}$, and $P_{j\mathrm{s}}(\hat{t}_j) = [P_{i\mathrm{s}}(t_i) = P_{\mathrm{s}}(t) =]\, P_{j\mathrm{s}}(t_j)$. Because $P_{j\mathrm{s}}(\hat{u}_j) = P_{i\mathrm{s}}(u_i) \neq \epsilon$, there exist $\alpha \in \Sigma^{\mathrm{s}}$ and $\hat{v}_j < \hat{u}_j$ such that $P_{j\mathrm{s}}(\hat{v}_j) = \epsilon$ and $\hat{v}_j\alpha \leq \hat{u}_j$. Then, $\hat{t}_j\hat{v}_j \in \overline{S_j}$, $P_{j\mathrm{s}}(\hat{t}_j\hat{v}_j) = [P_{j\mathrm{s}}(\hat{t}_j) =]\, P_{j\mathrm{s}}(t_j)$, and $\hat{t}_j\hat{v}_j\alpha \in \overline{S}_j$. Therefore, by the condition in Def. 6.1 we have that there exists $v_j \in (\Sigma^{\mathrm{p}}_j)^*$ such that $t_jv_j\alpha \in \overline{S_j}$. Note that, since $P_{j\mathrm{s}}(\hat{u}_j) \neq \epsilon$, there exist $\lambda \in \Sigma^{\mathrm{s}}$ and $v'_j \in \Sigma_j^*$ such that $\big(v'_j\lambda \leq \hat{u}_j$ and $P_{j\mathrm{s}}(v'_j\lambda) = P_{j\mathrm{s}}(\hat{u}_j)\big)$ $(*)$, i.e., there is an event λ which is the last shared event that occurs in $\hat{u}_j$. It may also be that $v'_j = \hat{v}_j$ and $\lambda = \alpha$. Nonetheless, if $P_{j\mathrm{s}}(\hat{u}_j) \neq \alpha$ — i.e., $\hat{v}_j < v'_j$ — note that we can reapply the same argument as before (where we use the unambiguity property) as many times as the length of $P_{j\mathrm{s}}(\hat{u}_j)$ minus 1 — i.e., in total, we apply the condition from Def. 6.1 as many times as there are shared events in u_i (or, equivalently, in $\hat{u}_j$). This implies that there exists $\tilde{v}_j \in \Sigma_j^*$ such that $P_{j\mathrm{s}}(\tilde{v}_j) = P_{j\mathrm{s}}(v'_j)$ and $t_j\tilde{v}_j\lambda \in \overline{S_j}$. From $(*)$, we have that there exists $w' \in (\Sigma^{\mathrm{p}}_j)^*$ such that $v'_j\lambda w' = \hat{u}_j$, which implies $\hat{t}_jv'_j\lambda w' = \hat{t}_j\hat{u}_j \in S_j$. But $P_{j\mathrm{s}}(\hat{t}_jv'_j\lambda w') = P_{j\mathrm{s}}(\hat{t}_jv'_j\lambda) = P_{j\mathrm{s}}(t_j\tilde{v}_j\lambda)$; thus, by the condition in Def. 6.2 there exists $w \in (\Sigma^{\mathrm{p}}_j)^*$ such that $t_j\tilde{v}_j\lambda w \in S_j$, implying $P_{j\mathrm{s}}(\tilde{v}_j\lambda w) = P_{j\mathrm{s}}(v'_j\lambda) = P_{j\mathrm{s}}(\hat{u}_j) = P_{i\mathrm{s}}(u_i)$, so $P_i^{-1}(u_i) \cap P_j^{-1}(\tilde{v}_j\lambda w) \neq \emptyset$ and hence there exists $u \in \Sigma^*$ such that $P_i(u) = u_i$ and $P_j(u) = \tilde{v}_j\lambda w$. Then, we have $P_i(tu) = t_iu_i \in S_i$ and $P_j(tu) = P_j(t_j\tilde{v}_j\lambda w) \in S_j$, which implies $tu \in S_1 \parallel S_2$ and so $t \in \overline{S_1 \parallel S_2}$. ■

Supplementary Remark 7: Suppose that a pair of automata $\mathbf{S}_i^0$ satisfying Cond. 1 is given as input for CBSS. Then, for the automata denoted by $\mathbf{S}_i$ and $\hat{\mathbf{S}}_i$ at any line of Proc. 4, we have that the pairs $(\mathbf{S}_1, \mathbf{S}_2)$ and $(\hat{\mathbf{S}}_1, \hat{\mathbf{S}}_2)$ satisfy Cond. 1.

Sup. Rem. 7 follows from Lem. 1, and Rem. 5 and 7.

Theorem 2. Consider a plant composed of two subsystems $\mathbf{M}_i$ with plantified specifications $\mathbf{K}_i$, as in Sec. III-A. Let $(\mathbf{S}_1, \mathbf{S}_2, v) = \mathrm{CBSS}(\mathbf{K}_1, \mathbf{K}_2)$. If v is true, let f_i be the local supervisors defined from $\mathbf{S}_i$ via Def. 5. Then, it holds that

1) $\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2) \in \mathcal{C}(K_1 \parallel K_2)$, and
2) $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2)}$,

where controllability is taken with respect to $\mathcal{L}(\mathbf{M}_1 \parallel \mathbf{M}_2)$ and $\Sigma^{\mathrm{p}}_{1,\mathrm{uc}} \cup \Sigma^{\mathrm{p}}_{2,\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$.

Proof:

First note the following two facts, so we can apply previous results from this paper. Firstly, the output of CBSS is also the

output of its last call of NEGOTIATION. Secondly, as $(\mathbf{K}_1, \mathbf{K}_2)$ satisfies Cond. 1 (Rem. 3), we can apply Sup. Rem. 7; so, for example, we have that the pair of automata outputted by $\mathrm{CBSS}(\mathbf{K}_1, \mathbf{K}_2)$ also satisfies this condition.

Thanks to TREATRESIDUAL and ENFORCERU, the automata outputted by CBSS are (i) trim, and its marked languages are (ii) unambiguous as stated in Def. 6. From (i) and (ii), by Prop. 1 we get that S_1 and S_2 are nonconflicting. This proves (2) by Cor. 2. Moreover, from nonconflict and (i), by Cor. 1 and Sup. Rem. 7 we get that $S_1 \parallel S_2 = \sup\mathcal{C}(\tilde{S}_1 \parallel \tilde{S}_2)$, where automata $\tilde{\mathbf{S}}_i$ are a pair that satisfies Cond. 1 and such that $\tilde{S}_i \subseteq K_i$. Thus, we have that $\sup\mathcal{C}(\tilde{S}_1 \parallel \tilde{S}_2) \subseteq \sup\mathcal{C}(K_1 \parallel K_2)$, so $S_1 \parallel S_2 \in \mathcal{C}(K_1 \parallel K_2)$. Finally, from (i) and Sup. Rem. 7, we can apply Lem. 2, which proves (1). ■

Theorem 3. Let $\mathbf{M}_i$ and $\mathbf{K}_i$ be subsystems and their plantified specifications, as in Sec. III-A, and $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{C}_1, \mathbf{C}_2) = \mathrm{NEGOTIATION}(\mathbf{K}_1, \mathbf{K}_2)$. If $\mathbf{S}_i$ are trim automata, with S_i unambiguous as stated in Def. 6, and if f_i is defined from $\mathbf{S}_i$ via Def. 5, we have that

1) $\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2) = \sup\mathcal{C}(K_1 \parallel K_2)$, and
2) $\mathcal{L}(f_1/\mathbf{M}_1) \parallel \mathcal{L}(f_2/\mathbf{M}_2) = \overline{\mathcal{L}_m(f_1/\mathbf{M}_1) \parallel \mathcal{L}_m(f_2/\mathbf{M}_2)}$,

where controllability is taken with respect to $\mathcal{L}(\mathbf{M}_1 \parallel \mathbf{M}_2)$ and $\Sigma^{\mathrm{p}}_{1,\mathrm{uc}} \cup \Sigma^{\mathrm{p}}_{2,\mathrm{uc}} \cup \Sigma^{\mathrm{s}}_{\mathrm{uc}}$.

Proof:

The assumption is that the automata $\mathbf{S}_i$ are (i) trim, and that its marked languages are (ii) unambiguous as stated in Def. 6.

By Lem. 1 and Rem. 3, we have that the pair of automata $\mathbf{S}_i$ satisfies Cond. 1. This, in addition to (i) and (ii), implies by Prop. 1 that S_1 and S_2 are nonconflicting, which proves (2) by Cor. 2. Moreover, from nonconflict and (i), by Cor. 1 we get that $S_1 \parallel S_2 = \sup\mathcal{C}(K_1 \parallel K_2)$. Finally, from (i), we can apply Lem. 2, which proves (1). ■

REFERENCES

[1] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming dr. frankenstein: Contract-based design for cyber-physical systems," *European journal of control*, vol. 18, no. 3, pp. 217–238, 2012.

[2] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, "Contracts for system design," *Foundations and Trends in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[3] R. Majumdar, K. Mallik, A. K. Schmuck, and D. Zufferey, "Assume-guarantee distributed synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3215–3226, 2020.

[4] W. Wonham and P. Ramadge, "Modular supervisory control of discrete-event systems," *Math. Control Signal Systems*, vol. 1, pp. 13–30, 1988.

[5] M. H. Queiroz and J. Cury, "Modular supervisory control of large scale discrete-event systems," in *Proc. 5th International Workshop on Discrete Event Systems (WODES'2000)*, Ghent, Belgium, 2000, pp. 103–110.

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

[7] K. Schmidt, T. Moor, and S. Perk, "Nonblocking hierarchical control of decentralized discrete event systems," *IEEE Transactions on Automatic Control*, vol. 53, no. 10, pp. 2252–2265, 2008.

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

[9] R. Su and J. Thistle, "A distributed supervisor synthesis approach based on weak bisimulation," in *2006 8th International Workshop on Discrete Event Systems*, 2006, pp. 64–69.

[10] K. Cai and W. M. Wonham, *Supervisor Localization, A Top-Down Approach to Distributed Control of Discrete-Event Systems*, ser. Lecture Notes in Control and Information Sciences. Springer, 2016.

[11] J. Komenda and T. Masopust, "Computation of controllable and coobservable sublanguages in decentralized supervisory control via communication," *Discrete Event Dynamic Systems*, vol. 27, 12 2017.

[12] K. Cai, R. Zhang, and W. M. Wonham, "Relative observability of discrete-event systems and its supremal sublanguages," *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 659–670, 2015.

[13] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discret. Event Dyn. Syst.*, vol. 17, no. 4, pp. 475–504, 2007.

[14] W. A. Apaza-Perez, C. Combastel, and A. Zolghadri, "On distributed symbolic control of interconnected systems under persistency specifications," *International Journal of Applied Mathematics and Computer Science*, vol. 30, no. 4, 2020.

[15] B. Finkbeiner and N. Passing, "Compositional synthesis of modular systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2021, pp. 303–319.

[16] A. M. Mainhardt and A.-K. Schmuck, "Assume-guarantee synthesis of decentralised supervisory control," *IFAC-PapersOnLine*, vol. 55, no. 28, pp. 165–172, 2022.

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

[18] K. Rudie and W. M. Wonham, "Think globally, act locally: Decentralized supervisory control," *IEEE Transactions on Automatic Control*, vol. 37, no. 11, 1992.

[19] S. Takai and T. Ushio, "Effective computation of an lm(g)-closed, controllable, and observable sublanguage arising in supervisory control," *Systems and Control Letters - SYST CONTROL LETT*, vol. 49, pp. 191–200, 07 2003.

[20] R. Malik, K. Akesson, H. Flordal, and M. Fabian, "Supremica–an efficient tool for large-scale discrete event systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5794–5799, 2017.

[21] R. Meira-Góes, A. Wintenberg, S. Matsui, and S. Lafortune, "MDESops: An open-source software tool for discrete event systems modeled by automata," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 6093–6098, 2023.

[22] J. Thistle, "Undecidability in decentralized supervision," *Systems & Control Letters*, vol. 54, no. 5, pp. 503–509, 2005.

[23] S. Tripakis, "Undecidable problems of decentralized observation and control on regular languages," *Information Processing Letters*, vol. 90, no. 1, pp. 21–28, 2004.

[24] A. M. Mainhardt, A. Wintenberg, S. Lafortune, and A.-K. Schmuck, "Formulating attacks with supervisory control," *IFAC-PapersOnLine*, vol. 58, no. 1, pp. 1–6, 2024.

Ana Maria Mainhardt received her B.S. degree in control and automation engineering at the Federal University of Santa Catarina (UFSC), Florianópolis, Brazil, in 2012. From the same university, she received her Master's degree in automation systems engineering, in 2015. Currently she is a PhD candidate at the Max-Planck Institute for Software Systems, Kaiserslautern, Germany, in the Cyber-Physical Systems group. Her ongoing research lies in the field of supervisory control theory, but her interests include privacy and security in discrete event systems and formal verification.

Anne-Kathrin Schmuck is an independent research group leader at the Max Planck Institute for Software Systems (MPI-SWS) in Kaiserslautern, Germany. Her group is externally funded by the Emmy Noether Programme of the German Science Foundation (DFG). She 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. Between 2015 and 2020 she was a postdoctoral researcher at MPI-SWS mentored by Rupak Majumdar. She currently serves as the co-chair of the IEEE CSS Technical Committee on Discrete Event Systems and as associate editor for the Springer Journal on Discrete Event Dynamical Systems, the IEEE Open Journal of Control Systems and the IFAC Journal on Nonlinear Analysis: Hybrid Systems. Her current research interests include cyber-physical system design, logical control software synthesis, reliability of automation systems and dynamical systems theory.