Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 388 (2007) 199–226

Theoretical Computer Science

www.elsevier.com/locate/tcs

Control of discrete-event systems with modular or distributed structure

Jan Komenda[a,*], Jan H. van Schuppen[b]

[a] *Institute of Mathematics, Czech Academy of Sciences, Žižkova 22, 616 62 Brno, Czech Republic*
[b] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

Received 28 August 2006; received in revised form 16 February 2007; accepted 6 July 2007

Communicated by M. Bonsangue and F. de Boer

Abstract

Most of the large scale state transition (also called discrete-event) systems are formed as parallel compositions of many small subsystems (modules). Control of modular and distributed discrete-event systems appears as an approach to handle computational complexity of synthesizing supervisory controllers for large scale systems. For both modular and distributed discrete-event systems sufficient and necessary conditions are derived for modular control synthesis to equal global control synthesis, while enforcing a safety specification in an optimal way (the language of the controlled system is required to be the supremal one achievable by an admissible controller and included in a safety specification language). The two cases of local (decomposable) and global (indecomposable) specifications are considered. The modular control synthesis has a much lower computational complexity than the corresponding global control synthesis for the respective sublanguages. The complexity is compared using explicit formulas.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Supervisory control; Modular discrete-event system; Distributed discrete-event system

1. Introduction

The purpose of the paper is to present an overview of recent results together with new results on control of modular (also called concurrent) and distributed discrete-event systems. Discrete-event systems (DES) are event-triggered dynamical systems which are studied in computer science with applications in manufacturing, communication networks, but also in software engineering (automated system design). In particular the various types of state transition systems (automata, Petri Nets, process algebras) are typical instances of DES. The topic of modular DES arises because of an increasing complexity of engineering systems, in particular of computer and communication networks. There is a strong need for system theoretical treatment of modular DES motivated by these emerging application fields. Control of discrete-event systems is a natural generalization of their verification that is now very well established for both finite and infinite-state transition systems.

* Corresponding author.
E-mail addresses: komenda@ipm.cz (J. Komenda), J.H.van.Schuppen@cwi.nl (J.H. van Schuppen).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.07.049

In computer science the problems of supervisory control synthesis are studied as automated synthesis. In control theory for DES the goal is not to verify the specification, but to impose it by means of a supervisor that runs in parallel with the original system. The supervisor is chosen such that the composed system meets the specification. In the Ramadge–Wonham framework it is an automaton which runs in parallel with the original system and the parallel composition of the original system with the supervisor (called closed-loop or controlled system) meets the specification given mostly by a language or a logical formula. In this way the specification is imposed on the controlled system. Game theoretic framework for control of DES is used in [26], where infinite behaviors and μ-calculus specifications are considered. In [14] branching time controllers that ensure bisimilarity of the controlled nondeterministic system with the specification are considered.

Although most of the verification and control problems for finite-state transition systems are decidable, the high complexity of most control and verification problems makes them practically difficult. Moreover, there are undecidable control problems for decentralized DES [15,23]. In order to limit the high computational complexity of (global) control synthesis efficient methods for component-based control synthesis are developed. Synthesis of modular and distributed systems has also been treated by computer scientists, see for example [17].

The novelty of the paper is in the following results: the necessary conditions for commutativity between parallel composition and supremal sublanguages (Theorems 4.7, 5.9 and Corollary 5.11), necessary conditions in the case of global (indecomposable) specifications (Theorems 6.7, 7.4 and Corollary 7.5), and necessary and sufficient conditions for commutativity between languages of the controlled systems with respect to the so-called antipermissive control policy in the case of global specification (Theorems 5.16 and 7.8). Sufficient conditions are derived using the coinductive proof principle. Our earlier results, where necessary and sufficient structural conditions for local to equal global computation of supremal sublanguages differ in case of modular and distributed DES with global specification languages, are sharpened by showing that the (weaker) necessary conditions of [7] and [8] are also sufficient.

Because this paper aims at readers from the theoretical computer science community, there is an additional tutorial text in Section 2 for computer scientists on the concepts and results of control theory for discrete-event systems. This paper is the extended version of the paper [11].

The paper has the following structure. The next section is an introduction to supervisory control. In Section 3 a coalgebraic approach used in the sufficiency proofs is briefly recalled. Section 4 is devoted to modular control with complete observations and with decomposable specification languages. In Section 5 the case of a distributed DES and a decomposable specification is treated. In Section 6 the case is discussed of a modular DES with an indecomposable specification. Finally in Section 7 the case of a distributed system with an indecomposable specification is treated. The complexity issues are discussed in Section 8. In Section 9 concluding remarks are proposed.

2. Control of discrete-event systems—Introduction

In this section basic notation and terminology of supervisory control is recalled. The notation used in this paper is mostly taken from the lecture notes of Wonham [27] and the book [2].

A (deterministic) *generator*

$$G = (Q, A, f, q_0, Q_m),$$

is an algebraic structure consisting of a *state set* Q, an *event set* A, a *partial transition function* $f : Q \times A \to Q$, an *initial state* $q_0 \in Q$, and a *subset of marked states* $Q_m \subseteq Q$. A transition is also denoted as $q \overset{a}{\mapsto} q^+ = f(q, a)$. If a transition is defined then this is denoted by $f(q, a)!$ Denote by A^* the set of all finite strings of elements of the alphabet A and the empty string ε. Extend the transition function f to $f : Q \times A^* \to Q$ by induction. Define respectively the *language* and the *marked language* of the generator as,

$$L(G) = \{s \in A^* | f(q_0, s)!\}, \quad L_m(G) = \{s \in L(G) | f(q_0, s) \in Q_m\}.$$

Note that unlike $L_m(G)$, $L(G)$ is always prefix-closed. The prefix closure of a language $K \subseteq A^*$ is denoted by prefix(K). We often abuse notation and write L instead of $L(G)$. The tuple of languages $(L_m(G), L(G))$ will be called the *behavior* of the generator. The system is said to be *nonblocking* if the prefix closure of the marked language $L_m(G)$ equals the language $L(G)$. This is equivalent to the property that every string of the system, $s \in L(G)$, can be extended to a marked string, thus there exists a string $v \in A^*$ such that $sv \in L_m(G)$.

A *controlled generator* is a structure

$$\begin{aligned} & (G, A_c, \Gamma_c), \text{ where,} \\ G \quad & \text{is a generator,} \\ A_c \subseteq A \quad & \text{is the subset of } \textit{controllable events}, \\ A_u = A \setminus A_c \quad & \text{is the subset of } \textit{uncontrollable events}, \text{ and} \\ \Gamma_c = \{S \subseteq A | A_u \subseteq S\}, & \text{ is called the } \textit{set of control patterns}. \end{aligned}$$

A *supervisory control* or a *supervisor* for the controlled generator is a map $S : L(G) \to \Gamma_c$. $S(s)$ is the subset of events that the supervisor enables after string s has been generated by G. The *closed-loop system* associated with a controlled generator and a supervisory control as denoted above is defined as the language $L(S/G) \subseteq A^*$ and the marked language $L_m(S/G) \subseteq L(S/G)$, which are the smallest sublanguages satisfying,

$$\begin{aligned} &(1) \quad \varepsilon \in L(S/G), \\ &(2) \quad \text{if } s \in L(S/G) \text{ and if } a \in S(s) \text{ such that } sa \in L(G) \\ &\qquad \text{then } sa \in L(S/G); \\ &L_m(S/G) = L(S/G) \cap L_m(G). \end{aligned}$$

Note that at the automata level the supervision is implemented by the parallel composition (synchronous product) of the generator and the supervisor. The supervisor must be such that if an uncontrollable event is defined in a state of the uncontrolled system then it must be defined also in the corresponding state of the supervisor so that this event is not disabled by synchronizing the uncontrolled system with the supervisor. This can be achieved via the so-called prioritized synchronous composition. Since we work with behaviors of deterministic automata, the original language framework introduced by W.M. Wonham is chosen.

Problem 2.1 (*Supervisory Control Problem*). Consider a controlled generator (G, A_c, Γ_c) and a specification sublanguage $K \subset L_m(G)$. Does there exist a supervisor S such that the closed-loop system satisfies (1) $L_m(S/G) = K$ and (2) $L(S/G)$ is nonblocking?

Because often the specification sublanguage K contains only the safe strings, thus the unsafe strings are excluded, the control objective (1) of the above problem is called the *safety control objective*. Not every language can be exactly achieved by a supervisor. The property called controllability is needed.

Definition 2.2. A language $K \subseteq A^*$ is said to be *controllable* with respect to plant language $L = L(G)$ and alphabet A_u if $\forall s \in \text{prefix}(K)$ and $\forall a \in A_u$ such that $sa \in L$ we have that $sa \in \text{prefix}(K)$. Equivalently, if

$$\text{prefix}(K) A_u \cap L \subseteq \text{prefix}(K). \tag{1}$$

Theorem 2.3. *(Due to Ramadge, Wonham, See [20].) There exists a nonblocking supervisory control S for a generator G such that $L_m(S/G) = K$ and $L(S/G) = \text{prefix}(K)$ if and only if*

(1) *K is controllable with respect to $L(G)$ and A_u and*
(2) *$K = \text{prefix}(K) \cap L_m(G)$ (then one says that K is $L_m(G)$-closed.).*

As a corollary for prefix-closed specifications:

Corollary 2.4. *Let $\emptyset \neq K \subseteq L_m(G)$ be prefix closed. There exists a supervisory control S for G such that $L_m(S/G) = K$ and $L(S/G) = K$ if and only if K is controllable with respect to $L(G)$ and A_u.*

The corresponding supervisory control $S : L(G) \to \Gamma_c$ is:

$$S(s) = A_u \cup \{a \in A_c : \ sa \in \text{prefix}(K)\}.$$

Note the abuse of notation, where the same symbol is used to denote a supervisor and a control law. This is justified by the fact that, considering a supervisor, the control law is described by the transition function of the supervisor in the form of the set of enabled active events after string s has been processed by the supervisor. Most often one is concerned only with the safety issue, i.e. the controlled behavior must be included in the specification language hence $L(S/G) \subseteq K$. This is why for specifications which are not controllable, supremal controllable sublanguages are considered. The notation $\sup \mathrm{C}(K, L, A_u)$ is chosen for the supremal controllable sublanguage of K with respect to L and A_u. This language always exists, it is the union of all controllable sublanguages because controllability is preserved by language unions [29].

A discrete-event system is said to have *complete observations* if all events are observed and are available for the supervisory control. A discrete-event system is said to have *partial observations* if only a strict subset of the events are observed and are available for the supervisory control. A *modular discrete-event system* is a system consisting of a composition of two or more subsystems where each subsystem or module has complete observations of its (local) events. A *distributed discrete-event system* is a system consisting of a composition of two or more subsystems where at least one subsystem has only partial observations of its events.

In the presence of partial observations additional issues appear. A *generator with partial observations* is a structure (G, A_o) where G is a generator, $A_o \subseteq A$ is called the subset of *observable events*, and $A_{uo} = A \setminus A_o$ is called the subset of *unobservable events*. In this case one defines the *natural projection* as morphism of monoids $P : A^* \to A_o^*$ such that $P(a) = P(\varepsilon) = \varepsilon$ for $a \in A_{uo}$ and $P(a) = a$ for $a \in A_o$. Hence, P erases the unobservable events. Note that the supervisor cannot distinguish between two strings with the same projections, i.e. after two such strings the same control law must be applied. Therefore, a supervisor with partial observations is a map $S : P(L(G)) \to \Gamma_c$. Define also the inverse projection $P^{-1} : \text{Pwr}(A_o^*) \to \text{Pwr}(A^*)$ on subsets of strings or languages.

Let K be a specification language. The supervisory control with partial observations is defined as:

$$S(s) = A_u \cup \{a \in A_c : \exists s' \in \text{prefix}(K) \text{ with } P(s') = s \text{ and } s'a \in \text{prefix}(K)\}. \tag{2}$$

This original control law is nowadays called permissive, because in order to allow an a-transition in G after string s is observed it is sufficient that there exists one indistinguishable string s' which can be prolongated by a within the specification, while there might exist another indistinguishable string s'' whose a-prolongation leaves the prefix-closure of K. In other words this permissive control policy yields in general a larger language than the specification, which is not desirable if safety properties are expressed by the specification. Therefore the dual control policy (called antipermissive) will be studied in the next sections. Note that at the automata level in the presence of partial observation only selfloops are allowed for supervisor's transitions labelled by unobservable events, because the set of enabled events, encoded by the supervisor's transition function, cannot change with an unobservable transition. Let us remark that the prefix-closed language $L(S/G) \subseteq A^*$ and the marked language $L_m(S/G) \subseteq L(S/G)$ are defined in the same way as above (for the case of complete observations) except that in the condition (2) for $sa \in L(S/G)$, i.e. for $a \in A$ to be enabled after $s \in L(G)$ has been generated, it is required that $a \in S(P(s))$ instead of $a \in S(s)$. More details can be found in the proof of the so-called controllability and observability theorem in [2], pages 192–194.

The additional property needed to exactly achieve a specification language by a supervisor with partial observations is called observability.

Definition 2.5. The sublanguage $K \subseteq L$ is said to be *observable* with respect to the plant language L and the projection P if

$$\begin{aligned} &\forall s, s' \in \text{prefix}(K),\ \forall a \in A_c,\\ &sa \in L,\ s'a \in \text{prefix}(K), \text{ and } P(s) = P(s') \ \Rightarrow\ sa \in \text{prefix}(K). \end{aligned} \tag{3}$$

Theorem 2.6 (*Due to F. Lin and W.M. Wonham, See [16]*). *Consider a generator with partial observations. There exists a nonblocking supervisory control S with partial observations such that* $L_m(S/G) = K$ *and* $L(S/G) = \text{prefix}(K)$ *if and only if*

(1) *K is controllable with respect to $L(G)$ and A_u,*
(2) *K is observable with respect to $L(G)$ and P, and*
(3) $K = \text{prefix}(K) \cap L_m(G)$. *($K$ is $L_m(G)$-closed.)*

Unfortunately, unlike controllability, observability is not preserved by language unions. This is why a stronger property, called normality, has been introduced.

Definition 2.7 (*[2]*). Consider a controlled generator with partial observations and a specification sublanguage $K \subseteq L_m(G)$. Call the specification sublanguage K (L, P)*-normal* if

$$\text{prefix}(K) = P^{-1}P(\text{prefix}(K)) \cap L. \tag{4}$$

It is known that normal languages are closed with respect to unions, hence the supremal normal sublanguage of K always exists, it is the union of all normal sublanguages of K and it is denoted by $\sup \mathrm{N}(K, L, P)$.

Recall finally that in the case $A_c \subseteq A_o$ normality coincides with observability. This assumption is widely accepted in the computer science community, where unobservable actions are supposed to be uncontrollable.

For control problems with partial observations and a safety control objective, *supremal controllable and normal sublanguages* are important.

Recall that the synchronous product (also called the parallel composition) of languages $L_1 \subseteq A_1^*$ and $L_2 \subseteq A_2^*$ is defined by $L = L_1 \parallel L_2 = \cap_{i=1}^{2} P_i^{-1}(L_i) \subseteq A^*$, where $P_i : A^* \to A_i^*,\ i = 1, 2$ are natural projections to the local event sets.

Definition 2.8. Consider two generators,

$$G_1 = (Q_1, A_1, f_1, q_{1,0}, Q_{1,m}),\ G_2 = (Q_2, A_2, f_2, q_{2,0}, Q_{2,m}).$$

Their *synchronous product* is the generator

$$\begin{aligned} G_1 \| G_2 &= (Q_1 \times Q_2, A_1 \cup A_2, f, q_0, Q_m), \\ q_0 &= (q_{1,0}, q_{2,0}),\quad Q_m = Q_{1,m} \times Q_{2,m}, \\ f((q_1, q_2), a) &= \begin{cases} (f_1(q_1, a), f_2(q_2, a)) & \text{if } a \in A_1 \cap A_2, \\ & f_i(q_i, a)! \text{ for } i = 1, 2, \\ (f_1(q_1, a), q_2), & \text{if } a \in A_1 \setminus A_2,\ f_1(q_1, a)! \\ (q_1, f_2(q_2, a)), & \text{if } a \in A_2 \setminus A_1\ f_2(q_2, a)! \\ \text{undefined}, & \text{otherwise.} \end{cases} \end{aligned}$$

It can then be proven that,

$$L(G_1 \| G_2) = L(G_1) \| L(G_2),\ L_m(G_1 \| G_2) = L_m(G_1) \| L_m(G_2).$$

Denote for $n \in \mathbb{Z}$ the set of the first n positive natural numbers by $\mathbb{Z}_{\mathrm{n}} = \{1, 2, \ldots, n\}$. Then Definition 2.8 can be extended from $n = 2$ to general $n \in \mathbb{N},\ n \geq 2$.

Definition 2.9. A *modular discrete-event system* (also called a *concurrent discrete-event system*) is the synchronous product of two or more *modules* or *local subsystems* in which each module has complete observations of the state of its own module but does not have observations of the states of other modules unless it shares events with these other modules. Mathematically, a modular discrete-event system with $n \in \mathbb{Z}$ modules is a structure $\{G_i, A_{i,c}, A_{i,o}, \Gamma_{i,c}, i \in \mathbb{Z}_{\mathrm{n}}\}$ consisting of n controlled generators. The associated *global system* is the synchronous product of the modules or the local subsystems, $\|_{i=1}^{n} G_i$. Denote the natural projections by $P_i : (\cup_{j=1}^{n} A_j)^* \to A_i^*$ for all $i \in \mathbb{Z}_{\mathrm{n}}$.

A *distributed discrete-event system* is a structure as above consisting of n controlled generators where at least one of the modules has only partial observations of that module.

3. Automata in terms of coalgebra

In this section generators introduced above are formulated in a coalgebraic framework. This was first done by J.J.M.M. Rutten, who called them *partial automata*, and his framework will be used in this paper. The transition function together with the output function can be viewed as a coalgebraic set functor.

Now we recall from [21] that partial automata are coalgebras of a special functor in the category of sets with functions as morphisms. Denote by $\Uparrow = \{\emptyset\}$ the one element set and by $2 = \{0, 1\}$ the set of Booleans.

A *partial automaton* is a pair $S = (S, \langle o, t \rangle)$, where S is a set of states, and $\langle o, t \rangle\ : S \to 2 \times (\Uparrow + S)^A$ is a pair of functions consisting of an output function $o\ : S \to 2$ and a transition function $t\ : S \to (\Uparrow + S)^A$. The output function o indicates whether a state $s \in S$ is accepting (or terminating): $o(s) = 1$, denoted also by $s \downarrow$, or not: $o(s) = 0$, denoted by $s \uparrow$. The transition function t associates with each state s in S a function $t(s) : A \to (\Uparrow + S)$. The set $\Uparrow + S$ is the disjoint union of S and $\Uparrow$. The meaning of the state transition function is that $t(s)(a) = \emptyset$ iff $t(s)(a)$ is undefined, which means that there is no a-transition from the state $s \in S$. $t(s)(a) \in S$ means that the a-transition from s is possible and we define in this case $t(s)(a) = s_a$, which is denoted mostly by $s \overset{a}{\to} s_a$. This notation can be extended by induction to arbitrary strings in A^*. Assuming that $s \overset{w}{\to} s_w$ has been defined, define $s \overset{wa}{\to}$ iff $t(s_w)(a) \in S$, in which case $s_{wa} = t(s_w)(a)$, also denoted by $s \overset{wa}{\to} s_{wa}$. It is easy to see that partial automata are coalgebras of the set functor $F = 2 \times (\Uparrow + (.)\ )^A$.

A *homomorphism* between partial automata $S = (S, \langle o, t\rangle)$ and $S' = (S', \langle o', t'\rangle)$ is a function $f : S \to S'$ with, for all $s \in S$ and $a \in A$:

$$o'(f(s)) = o(s) \text{ and } s \overset{a}{\to} s_a \text{ iff } f(s) \overset{a}{\to} f(s_a),$$

in which case: $f(s)_a = f(s_a)$.

$$\begin{tikzcd}
(\Uparrow + S)^A \arrow[dd, "(1+f)^A"] & S \arrow[l, "t"'] \arrow[dd, "f"'] \arrow[dr, "o"] & \\
 & & 2 \\
(\Uparrow + S')^A & S' \arrow[l, "t'"] \arrow[ur, "o'"] &
\end{tikzcd}$$

A partial automaton $S' = (S', \langle o', t'\rangle)$ is a *subautomaton* of $S = (S, \langle o, t\rangle)$ if $S' \subseteq S$ and the inclusion function $i : S' \to S$ is a homomorphism.

A *bisimulation* between two partial automata $S = (S, \langle o, t\rangle)$ and $S' = (S', \langle o', t'\rangle)$ is a relation $R \subseteq S \times S'$ with, for all $s \in S$ and $s' \in S'$:

$$\text{if } \langle s, s'\rangle \in R \text{ then } \begin{cases} \text{(i) } o(s) = o(s'), \text{ i.e. } s\downarrow \text{ iff } s'\downarrow \\ \text{(ii) } \forall a \in A:\ s \overset{a}{\to} \Rightarrow (s' \overset{a}{\to} \text{ and } \langle s_a, s'_a\rangle \in R,) \text{ and} \\ \text{(iii) } \forall a \in A:\ s' \overset{a}{\to} \Rightarrow (s \overset{a}{\to} \text{ and } \langle s_a, s'_a\rangle \in R). \end{cases}$$

We write $s \sim s'$ whenever there exists a bisimulation R with $\langle s, s'\rangle \in R$. This relation is the union of all bisimulations, i.e. the greatest bisimulation also called bisimilarity.

3.1. Final automaton of partial languages

Below a partial automaton of partial languages over an alphabet (input set) A, denoted by $\mathcal{L} = (\mathcal{L}, \langle o_{\mathcal{L}}, t_{\mathcal{L}}\rangle)$, is recalled. More formally,

$$\mathcal{L} = \{(V, W) \mid V \subseteq W \subseteq A^*,\ W \neq \emptyset, \text{ and } W \text{ is prefix-closed}\},$$

and the transition function $t_{\mathcal{L}} : \mathcal{L} \to (\Uparrow + \mathcal{L})^A$ is defined using input derivatives. Recall that for any partial language $L = (L^1, L^2) \in \mathcal{L}$, $L_a = (L^1_a, L^2_a)$, where $L^i_a = \{w \in A^* \mid aw \in L^i\}$, $i = 1, 2$. If $a \notin L^2$ then L_a is undefined. Given any $L = (L^1, L^2) \in \mathcal{L}$, the partial automaton structure of $\mathcal{L}$ is given by:

$$o_{\mathcal{L}}(L) = \begin{cases} 1 & \text{if } \varepsilon \in L^1 \\ 0 & \text{if } \varepsilon \notin L^1 \end{cases} \text{ and } t_{\mathcal{L}}(L)(a) = \begin{cases} L_a & \text{if } L_a \text{ is defined} \\ \emptyset & \text{otherwise.} \end{cases}$$

Notice that if L_a is defined, then $L^1_a \subseteq L^2_a$, $L^2_a \neq \emptyset$, and L^2_a is prefix-closed. The following notational conventions will be used: $L\downarrow$ iff $\varepsilon \in L^1$, and $L \overset{w}{\to} L_w$ iff L_w is defined (iff $w \in L^2$).

3.2. Final automata and coinduction

We recall from [21] that the partial automaton $\mathcal{L}$ is final among all partial automata.

Theorem 3.1. *The partial automaton $\mathcal{L} = (\mathcal{L}, \langle o_{\mathcal{L}}, t_{\mathcal{L}}\rangle)$ is final among all partial automata: for any partial automaton $S = (S, \langle o, t\rangle)$ there exists a unique homomorphism $l : S \to \mathcal{L}$. This homomorphism identifies bisimilar states: for $s, s' \in S$: $l(s) = l(s')$ iff $s \sim s'$.*

As a consequence, $\mathcal{L}$ satisfies the coinductive proof principle.

Theorem 3.2. *$\mathcal{L}$ satisfies the principle of coinduction: for all K and L in $\mathcal{L}$, if $K \sim L$ then $K = L$.*

Another consequence of finality of $\mathcal{L}$ is that coinductive definitions can be used: definition by coinduction of an operation on partial languages is done by defining the same coalgebraic structure on this operation: i.e. input derivatives and output functions. The coinductive definition of supremal controllable sublanguage and synchronous product will be needed. For the synchronous product we assume that K is defined over the alphabet A_1 and L over A_2. Then the synchronous product $K \parallel L$ is a partial language over $A_1 \cup A_2$ with the following coinductive definition:

Definition 3.3 (*Synchronous Product*).

$$(K \parallel L)_a = \begin{cases} K_a \parallel L_a & \text{if } a \in A_1 \cap A_2 \\ K_a \parallel L & \text{if } a \in A_1 \setminus A_2 \\ K \parallel L_a & \text{if } a \in A_2 \setminus A_1 \end{cases}$$

and $(K \parallel L) \downarrow$ if and only if $K \downarrow$ and $L \downarrow$.

In the definitions below $A_u \subseteq A$ denotes the subset of the uncontrollable events. Now we recall from [10] the following binary operation on partial languages:

Definition 3.4 (*Supremal Controllable Sublanguage*). Define the following binary operation on (partial) languages for all $K, L \in \mathcal{L}$ and $\forall a \in A$:

$$(K/_{A_u}^{SC} L)_a = \begin{cases} K_a/_{A_u}^{SC} L_a & \text{if } K \xrightarrow{a} \text{ and } L \xrightarrow{a} \\ & \text{and if } \forall u \in A_u^* : \\ & L_a \xrightarrow{u} \Rightarrow K_a \xrightarrow{u} \\ \emptyset & \text{otherwise} \end{cases}$$

and $(K/_{A_u}^{SC} L) \downarrow$ iff $L \downarrow$.

We have shown in [10] that for a partial order that considers only second (prefix-closed) components of the languages involved:

Theorem 3.5.

$$\begin{aligned} (K/_{A_u}^{SC} L) &= \sup \mathrm{C}(K, L, A_u) & (5) \\ &= \sup\{M \subseteq K \mid M \textit{is controllable with respect to } L,\ A_u\}, & (6) \end{aligned}$$

i.e. $K/_{A_u}^{SC} L$ *equals the supremal controllable sublanguage of* K.

More details about coinduction and finality can be found in [22] or [21].

4. Modular supervisory control with a decomposable specification

In this section the concurrent behavior of the modules $\{G_i, i \in \mathbb{Z}_n\}$ is considered. This case arises in applications and was often studied from the late 1980s: e.g. [30,28,25,3,18,19] etc. In the first papers on the topic, the input alphabets of all local components were identical [30]. The general case of different local input alphabets has been studied in [28], where shared events are supposed to be controllable. This assumption has been generalized recently in [25] to the condition that the shared events must have the same control status for all subsystems that share a particular event. Very little attention has been payed so far to the distributed control. A special case of it was studied in [18].

Consider the local alphabets of these subplants, $\{A_i, i \in \mathbb{Z}_n\}$, which are not necessarily pairwise disjoint. Denote the partition of the local alphabet into the subset of controllable events, A_{ic}, and the subset of uncontrollable events, A_{iu}, by $A_i = A_{iu} \cup A_{ic}$ for all $i \in \mathbb{Z}_n$.

Definition 4.1. Consider a modular DES. The local plants $\{G_i, i \in \mathbb{Z}_n\}$ *agree on the controllability of their common events* if

$$A_{iu} \cap A_j = A_i \cap A_{ju},\ \forall i, j \in \mathbb{Z}_n. \tag{7}$$

This definition stemming from [25] means that the events shared by two modules or local subsystems must have the same control status for both controllers associated with these subsystems. In the following it will often be assumed that the modules satisfy the condition of agreement on the controllability of their common events. Denote $A_c = \cup_{i=1}^n A_{ic}$.

The assumption on the agreement on common events implies that $A_{ic} = A_c \cap A_i$. Also, if we denote $A_u = \cup_{i=1}^n A_{iu}$ then we still have the disjoint union $A = A_c \cup A_u$ due to the assumption of agreement on the controllability of their common events. Denote by $A = \cup_{i=1}^n A_i$ the global alphabet and by $P_i : A \to A_i$ the projections to the local alphabets. The concept of inverse projection $P_i^{-1} : \mathrm{Pwr}(A_i) \to \mathrm{Pwr}(A)$ is also used.

The local plant languages or the languages of the modules will be denoted by $\{L_i, i \in \mathbb{Z}_n\}$ and the local specification languages by $\{K_i, i \in \mathbb{Z}_n\}$. We assume in this section that the global plant L and the specification K languages are decomposable into local plant and local specification languages: $L = \|_{i=1}^n L_i$ and $K = \|_{i=1}^n K_i$. This formulation is equivalent to the following definition.

Definition 4.2. Consider a modular DES. One says that $L \subseteq A^*$ is *decomposable* with respect to $\{P_i,\ i \in \mathbb{Z}_n\}$ if $L = \|_{i=1}^n P_i(L)$.

Proposition 4.3. *Consider a modular DES.* $L \subseteq A^*$ *is decomposable with respect to* $\{P_i,\ i \in \mathbb{Z}_n\}$ *if and only if there exists* $\{L_i \subseteq A_i^*, i \in \mathbb{Z}_n\}$ *such that*

$$L = \|_{i=1}^n L_i = \cap_{i=1}^n (P_i)^{-1}(L_i). \tag{8}$$

Proof. ($\Rightarrow$) It is sufficient to consider $L_i := P_i(L)$.
($\Leftarrow$) If $L = \|_{i=1}^n L_i = \cap_{i=1}^n (P_i)^{-1}(L_i)$, then it follows from properties of projections that

$$P_i(L) \subseteq L_i \cap \cap_{j \neq i} P_i P_j^{-1}(L_j) \subseteq L_i,\ \forall i \in \mathbb{Z}_n.$$

Thus we have that,

$$\|_{i=1}^n P_i(L) = \cap_{i=1}^n (P_i)^{-1} P_i(L) \subseteq \cap_{i=1}^n P_i^{-1}(L_i) = L,$$

by our assumption. The first inclusion follows from the fact that $P_i(P_i)^{-1}$ is identity and that projection of an intersection is contained in the intersection of projections. The inclusion $L \subseteq \cap_{i=1}^n P_i^{-1}(P_i(L))$ is obvious. □

Note that $P_j(L) \subseteq L_j$ for all $j \in \mathbb{Z}_n$ means that for any tuple of languages $\{L_i' \subseteq A_i^*,\ i \in \mathbb{Z}_n\}$ such that $L = \|_{i=1}^n L_i'$ we have $P_j(L) \subseteq L_j'$ for all $j \in \mathbb{Z}_n$, i.e. $\{P_i(L),\ i \in \mathbb{Z}_n\}$ is the smallest possible decomposition of L into local languages.

Definition 4.4. Consider a modular discrete-event system and either a global specification language or a family of local specifications. From the local specifications one can always compute the global specification as described above.

Global control synthesis of a modular discrete-event system is the procedure by which first all modules are combined into the global plant and then control synthesis is carried out as described in Section 2. This can refer to either computation of a supervisor which meets the specification or to the computation of the supremal (safe) supervisor.

Modular control synthesis of a modular discrete-event system is the procedure by which control synthesis is carried out for each module or local subsystem. The global supervisor formally consists of the synchronous product of the local supervisors though that product is not computed in practice.

In terms of behaviors, the optimal global control synthesis is represented by the closed-loop language

$$\sup \mathrm{C}(K, L, A_u) = \sup \mathrm{C}(\|_{i=1}^n K_i, \|_{i=1}^n L_i, A_u),$$

using the operation supremal controllable sublanguage $\sup \mathrm{C}(K, L, A_u)$ defined in the last sections. Similarly, modular control synthesis yields in terms of behaviors

$$\|_{i=1}^n \sup \mathrm{C}(K_i, L_i, A_{iu}).$$

Problem 4.5. Consider a modular discrete-event system and a decomposable specification language. Determine necessary and sufficient conditions with respect to which modular control synthesis equals global control synthesis for the supremal controllable sublanguage within the specification language. Equivalently,

$$\|_{i=1}^n \sup \mathrm{C}(K_i, L_i, A_{iu}) = \sup \mathrm{C}(\|_{i=1}^n K_i, \|_{i=1}^n L_i, A_u). \tag{9}$$

Later in the paper also the problems are investigated of when modular control synthesis equals global control synthesis for the supremal normal sublanguage, for the supremal controllable and normal sublanguage, and for the closed-loop language in case of an antipermissive control policy.

There exists an example which establishes that modular control synthesis does not equal global control synthesis in general. Theorem 4.7 provides necessary and sufficient conditions for modular control synthesis to equal global control synthesis. This problem has been studied algebraically in [25]. The concept of mutual controllability [25] plays the key role.

Definition 4.6. Consider a modular DES. The modular plant languages $\{L_i \subseteq A_i^*,\ i \in Z_n\}$ are called *mutually controllable* if

$$\text{prefix}(L_j)(A_{ju} \cap A_i) \cap P_j(P_i)^{-1}\text{prefix}(L_i) \subseteq \text{prefix}(L_j), \forall i, j \in \mathbb{Z}_\text{n},\ i \neq j. \tag{10}$$

Note that both local and global plant languages are typically prefix closed, in which case the prefix closures above can be removed.

Mutual controllability can be viewed as local controllability of the modular plant languages with respect to the shared uncontrollable events; thus, for all $i, j \in \mathbb{Z}_\text{n}$ with $i \neq j$, the modular language L_j is controllable with respect to the shared uncontrollable events $(A_{ju} \cap A_i)$ and the local view of the other module $(P_i(P_j)^{-1}(L_j))$ as the new plant. This condition is important for modular computation of global supremal controllable sublanguages.

The computational complexity of checking mutual controllability is much lower than that of checking controllability of a sublanguage over the global alphabet.

Theorem 4.7. *Modular control synthesis equals global control synthesis for the supremal controllable sublanguage in the case of a modular DES. Assume that the local plants agree on the controllability of their common events.*

If the local plant languages $\{L_i \subseteq A_i^*,\ i \in Z_n\}$ are mutually controllable then for any decomposable specification $K \subseteq L$

$$\|_{i=1}^n \sup \mathrm{C}(K_i, L_i, A_{iu}) = \sup \mathrm{C}(\|_{i=1}^n K_i, \|_{i=1}^n L_i, A_u). \tag{11}$$

Conversely, if for fixed local plant languages $\{L_i,\ i \in Z_n\}$ Eq. (11) holds for any $\{K_i \subseteq L_i, i \in \mathbb{Z}_\text{n}\}$ then, for all $i \in \mathbb{Z}_\text{n}$, $P_i(L)$ is controllable with respect to L_i and A_{iu}; equivalently, then

$$P_i(L)A_{iu} \cap L_i \subseteq P_i(L), \forall i \in \mathbb{Z}_\text{n}. \tag{12}$$

Proof. The sufficiency part is due to K.C. Wong and S.-H. Lee, see [25]. We have proposed a coalgebraic proof in [4] and [9].

It remains to prove the necessity part of the theorem. Assume now that for fixed local plant languages $\{L_i, i \in \mathrm{Z}_n\}$ Eq. (11) holds for any local specification languages $K_i \subseteq L_i$. Then it holds in particular for $K_i := P_i(L)$. Since L and K are decomposable we have

$$L = \bigcap_{i=1}^n P_i^{-1} P_i(L) = \bigcap_{i=1}^n P_i^{-1} K_i = K, \text{ (cf. Proposition 4.3)}. \tag{13}$$

$$\begin{aligned} L &= K = \sup \mathrm{C}(L, L, A_u) = \|_{i=1}^n \sup \mathrm{C}(P_i(L), L_i, A_{iu}) \\ &= \cap_{i=1}^n P_i^{-1}(\sup \mathrm{C}(P_i(L), L_i, A_{iu})), \\ L &\subseteq (P_i)^{-1}(\sup \mathrm{C}(P_i(L), L_i, A_{iu}), \forall i \in \mathbb{Z}_\text{n}; \end{aligned} \tag{14}$$

Note that $P_i(L) \subseteq L_i, \forall i \in \mathbb{Z}_\text{n}$, because L is decomposable. Indeed

$$P_i(L) = P_i\left(\bigcap_{i=1}^n P_i^{-1}(L_i)\right) \subseteq P_i(P_i)^{-1}(L_i) \cap \cap_{j\neq i} P_i(P_j)^{-1}(L_j) \subseteq L_i,$$

where the last inclusion follows from $P_i(P_i)^{-1}(L_i) = L_i$.

By applying the projection on both sides of inclusion (14)

it follows from the monotonicity of projections that

$$P_i(L) \subseteq P_i(P_i)^{-1}\sup \mathrm{C}(P_i(L), L_i, A_{iu}) = \sup \mathrm{C}(P_i(L), L_i, A_{iu}) \subseteq P_i(L),$$

from the definition of supremal controllable sublanguages. Hence,

$$P_i(L) = \sup \mathrm{C}(P_i(L), L_i, A_{iu}),$$

i.e. for all $i \in \mathbb{Z}_\text{n}$, $P_i(L)$ is controllable with respect to L_i and A_{iu}. □

Note that in the case, where for all $i \in \mathbb{Z}_n$ $P_i(L) = L_i$ the necessary condition is trivially satisfied. See Section 8.1 for the comparison of the computational time complexity of global and modular control synthesis for the supremal controllable sublanguage in the case of a decomposable specification.

Finally we show that mutual controllability is not a necessary structural condition, which is illustrated by the following example, where the modular and global computation of supremal controllable sublanguage coincide for all specifications, while mutual controllability does not hold.

Example 4.8. Let $A_1 = \{a, u, u_1\}$ and $A_2 = \{a, u, u_2\}$ with $A_{1u} = \{u, u_1\}$ and $A_{2u} = \{u, u_2\}$. Consider the local plant languages $L_1 = \text{prefix}(auu_1u^*)$ and $L_2 = \text{prefix}(au^*u_2)$. Then for any local specifications $K_i \subseteq L_i$ and $i = 1, 2$ we have $\sup \mathrm{C}(K_i, L_i, A_{iu}) = \{\varepsilon\}$ as well as $\sup \mathrm{C}(\|_{i=1}^n K_i, \|_{i=1}^n L_i, A_u) = \varepsilon$, i.e. Eq. (11) holds trivially true, while L_1 and L_2 are not mutually controllable. Indeed, $auu \in L_1(A_{1u} \cap A_2) \cap P_1 P_2^{-1}(L_2) \setminus L_1$, which contradicts mutual controllability. This shows that mutual controllability is not a necessary structural (specification independent) condition.

5. Distributed supervisory control with a decomposable specification

In this section we consider the situation of a distributed plant, thus for which at least one module or subsystem does not have complete observations but only partial observations. For each local plant, the local alphabet admits a partition, denoted by $A_i = A_{o,i} \cup A_{uo,i}$, $\forall i \in \mathbb{Z}_n$ into locally observable and locally unobservable event sets. The global system has observation set $A_o = \cup_{i=1}^n A_{o,i} \subseteq A = \cup_{i=1}^n A_i$. Globally unobservable events are denoted by $A_{uo} = A \setminus A_o$ and locally unobservable events by $A_{uo,i} = A_i \setminus A_{o,i}$. The projections of the global alphabet into the local ones are denoted by $P_i : A^* \to A_i^*$, $i \in \mathbb{Z}_n$. Partial observations in individual modules are expressed via local projections $P_i^{\text{loc}} : A_i^* \to A_{o,i}^*$, while the global projection is denoted by $P : A^* \to A_o^*$.

Definition 5.1. Consider a distributed DES. The local plants are said to *agree on the observability of their common events* if

$$A_{o,i} \cap A_j = A_i \cap A_{o,j}, \forall i, j \in \mathbb{Z}_n. \tag{15}$$

5.1. Supremal controllable and normal sublanguages

In this subsection we recall a single step algorithm for computation of supremal controllable and normal sublanguages from [10]. The concept of ε-transitions is needed to formulate an observational indistinguishability relation due to partial observations. Generators of DES are partial automata $S = (S, \langle o, t \rangle)$.

Definition 5.2 (*ε-Transition*). For $s \in S$ we define $s \overset{\varepsilon}{\Rightarrow} s'$ if $\exists \tau \in A_{uo}^* : s \overset{\tau}{\to} s_\tau = s'$.

Denote the initial state of the DES generator S by s_0. An auxiliary concept that reflects the fact which is due to partial observations and it is not possible to distinguish between states is recalled from [10]:

Definition 5.3 (*Observational Indistinguishability Relation on S*). A binary relation $\text{Aux}(S)$ on S, called the *observational indistinguishability relation* is the smallest relation satisfying:

(i) $\langle s_0, s_0 \rangle \in \text{Aux}(S)$
(ii) $\forall \langle s, t \rangle \in \text{Aux}(S) : (s \overset{\varepsilon}{\Rightarrow} s' \text{ and } t \overset{\varepsilon}{\Rightarrow} t') \Rightarrow \langle s', t' \rangle \in \text{Aux}(S)$
(iii) $\forall \langle s, t \rangle \in \text{Aux}(S)$ and $\forall a \in A_o$: $(s \overset{a}{\to} s_a \text{ and } t \overset{a}{\to} t_a) \Rightarrow \langle s_a, t_a \rangle \in \text{Aux}(S)$.

Next we recall the concept of state-partition automaton, which we define here as follows.

Definition 5.4 (*State-Partition Automaton*). A partial automaton S is called a state-partition automaton if $\forall s \in S$: $s = (s_0)_{w_1} = (s_0)_{v_1}$ for $w_1 \in A^*$ and $v_1 \in A^*$ and $\forall s' \in S$: $s' = (s_0)_{v_2}$ for $v_2 \in A^*$ with $P(v_2) = P(v_1)$ there exists $w_2 \in A^*$ with $P(w_2) = P(w_1)$ and $s' = (s_0)_{w_2}$.

Since for $P(w_1) = P(v_1)$ it is sufficient to put $w_2 = v_2$, it is equivalent to require the condition of Definition 5.4 to hold only for $P(w_1) \neq P(v_1)$.

We assume that a partial language L is represented by a partial automaton S with initial state s_0. We mean by this that the corresponding behavior homomorphisms $l : S \to \mathcal{L}$ satisfies $l(s_0) = L$. We recall [10]:

Lemma 5.5. *For any $s, s' \in S$: $\langle s, s'\rangle \in \mathrm{Aux}(S)$ if and only if there exists $w, w' \in L$ such that $P(w) = P(w')$, $s = (s_0)_w$ and $s' = (s_0)_{w'}$. Moreover, if S is a state-partition automaton then $\forall v \in L$ and $s' \in S$ we have $\langle (s_0)_v, s'\rangle \in \mathrm{Aux}(S)$ if and only if there exists $v' \in L$ such that $P(v) = P(v')$ and $s' = (s_0)_{v'}$.*

Remark 5.6. Since we deal with properties that depend only on prefix-closed languages we write in the sequel K and L instead of prefix(K) and prefix(L).

Now we repeat a 'single-step' algorithm for the computation of supremal controllable and normal sublanguages from [10] that will form the basis for our main results.

Algorithm 1. Let automata S and T representing K and L (with $K \subseteq L$), respectively, be such that S is a subautomaton of T and S is a state-partition automaton. The transition functions of S and T are denoted by $\rightarrow$ and $\rightarrow_1$, respectively. Let us construct the partial automaton $\tilde{S} = (\tilde{S}, \langle \tilde{o}, \tilde{t}\rangle)$ with $\tilde{t}$ denoted by $\rightarrow_{'}$.

Define the auxiliary condition (*) as follows:
if $a \in A_u \cup A_{uo}$ then $\forall u \in (A_u \cup A_{uo})^*$: $s_a \overset{u}{\rightarrow} \Rightarrow\ s_a \overset{u}{\rightarrow}_1$;
if $a \in A_c \cap A_o$ then $\forall s' \approx_{\mathrm{Aux}(S)} s :\ s' \overset{a}{\rightarrow} \Rightarrow\ s' \overset{a}{\rightarrow}_1$, in which case also $\forall u \in (A_u \cup A_{uo})^*$: $s'_a \overset{u}{\rightarrow} \Rightarrow\ s'_a \overset{u}{\rightarrow}_1$.

(1) Let $\tilde{S} := \{s_0\}$.
(2) For any $s \in \tilde{S}$ and $a \in A$ we put $s \overset{a}{\rightarrow}_{'} s_a$ if $s \overset{a}{\rightarrow}_1$ and condition (*) is satisfied; and we put in the case $s \overset{a}{\rightarrow}_{'}$ also $\tilde{S} := \tilde{S} \cup \{s_a\}$.
(3) For any $s \in \tilde{S}$ we put $\tilde{o}(s) = o(s)$.

As usual, we denote by $\tilde{l}$ the unique (behavior) homomorphism given by finality of $\mathcal{L}$.

We can verify by coinduction that [10]:

Theorem 5.7. *$\tilde{l}(s_0)$ is the supremal controllable (with respect to L and A_u) and (L, P)-normal sublanguage of K.*

5.2. Modular and global control synthesis for distributed DES

The problem is to determine necessary and sufficient conditions for modular control synthesis to equal global control synthesis for a distributed control system. In Theorem 5.9 and Corollary 5.11 below a condition similar to mutual controllability is needed. By analogy it is called mutual normality.

Definition 5.8. Consider a distributed DES. Local plant languages $\{L_i \subseteq A_i^*,\ i \in \mathbb{Z}_n\}$ are called *mutually normal* if

$$(P_i^{\mathrm{loc}})^{-1} P_i^{\mathrm{loc}}(L_i) \cap P_i(P_j)^{-1}(L_j) \subseteq L_i,\ \forall i, j \in \mathbb{Z}_n,\ i \neq j. \tag{16}$$

Mutual normality can be viewed as normality of the local plant languages with respect to the local views of the other plant languages. Let us introduce the notation $\sup \mathrm{CN}(K, L, P, A_u)$ for the supremal (L, P)-normal and controllable sublanguage of K with respect to A_u.

Using Algorithm 1 we have proven in [5] the sufficiency part of the following theorem:

Theorem 5.9. *Distributed control synthesis equals global control synthesis for supremal controllable and normal sublanguage in the case of a distributed DES. Assume that the local plants agree on the controllability of their common events and on the observability of their common events.*

If the local plant languages $\{L_i \subseteq A_i^*,\ i \in Z_n\}$ are mutually controllable and mutually normal then for any decomposable specification $K \subseteq L$

$$\|_{i=1}^{n} \sup \mathrm{CN}(K_i, L_i, P_i^{\mathrm{loc}}, A_{iu}) = \sup \mathrm{CN}(\|_{i=1}^{n} K_i, \|_{i=1}^{n} L_i, P, A_u). \tag{17}$$

Conversely, if for fixed local plant languages $\{L_i\ \ i \in Z_n\}$ Eq. (17) holds for any $\{K_i \subseteq L_i, i \in \mathbb{Z}_n\}$, then $\{P_i(L),\ i \in \mathbb{Z}_n\}$ is such that (1) $P_i(L)$ is normal with respect to L_i and P_i^{loc} for all $i \in \mathbb{Z}_n$ and (2) $P_i(L)$ is controllable with respect to L_i and A_{iu} for all $i \in \mathbb{Z}_n$.

Proof. The complete sufficiency proof is available in [5]. Now we show the necessity part of the theorem. Assume now that for fixed local plant languages $\{L_i \; i \in \mathbb{Z}_n\}$ Eq. (17) holds for any local specification languages $K_i \subseteq L_i$. Then it holds in particular for $K_i := P_i(L)$. Since L is decomposable we have

$$L = \bigcap_{i=1}^{n} P_i^{-1} P_i(L) = K;$$

$$L = \sup \mathrm{CN}(L, L, P, A_u) = \|_{i=1}^{n} \sup \mathrm{CN}(P_i(L), L_i, P_i^{\mathrm{loc}}, A_{iu}). \text{ Thus,}$$

$$L \subseteq (P_i)^{-1} \sup \mathrm{CN}(P_i(L), L_i, P_i^{\mathrm{loc}}, A_{iu}), \forall i \in \mathbb{Z}_n. \text{ Recall that} \tag{18}$$

$$P_i(L) \subseteq L_i, \text{ because } L \text{ is decomposable.} \tag{19}$$

By applying projection P_i on both sides of the inclusion (18)

it follows from the monotonicity of projections that

$$P_i(L) \subseteq P_i(P_i)^{-1} \sup \mathrm{CN}(P_i(L), L_i, P_i^{\mathrm{loc}}, A_{iu}) = \sup \mathrm{CN}(P_i(L), L_i, P_i^{\mathrm{loc}}, A_{iu}).$$

Also, $\sup \mathrm{CN}(P_i(L), L_i, P_i^{\mathrm{loc}}, A_{iu}) \subseteq P_i(L)$,

from the definition of supremal normal sublanguages. Hence,

$$P_i(L) = \sup \mathrm{CN}(P_i(L), L_i, P_i^{\mathrm{loc}}, A_{iu}),$$

which means that for all $i \in \mathbb{Z}_n$, $P_i(L)$ is controllable with respect to L_i and A_{iu} and $P_i(L)$ is normal with respect to L_i and P_i^{loc}. □

In [25] there is a procedure to change a plant which does not satisfy the mutual controllability condition to one that satisfies it. It may be that a similar procedure can be found in the future for mutual normality. Nevertheless one cannot hope to find a universal procedure how to make a set of local plant languages mutually normal. Indeed, in the shuffle case mutual normality cannot hold as we show in the next section. However this is a much simpler case, where mutual normality is not needed. In fact, we have

Corollary 5.10. *Assume that the local alphabets are pairwise disjoint, i.e. $A_i \cap A_j = \emptyset$ for any $i, j \in \mathbb{Z}_n$ with $i \neq j$. Then*

$$\|_{i=1}^{n} \sup \mathrm{CN}(K_i, L_i, P_i^{\mathrm{loc}}, A_{iu}) = \sup \mathrm{CN}(\|_{i=1}^{n} K_i, \|_{i=1}^{n} L_i, P, A_u). \tag{20}$$

Proof. Note that in view of Theorem 5.13 (the shuffle case) mutual controllability is trivially satisfied and mutual normality is not needed for this sufficiency condition. □

In the case where controllability is not an issue we obtain the following corollary.

Corollary 5.11. *Distributed control synthesis equals global control synthesis for supremal normal sublanguages in the case of a distributed DES. Assume that the local plants agree on the observability of their common events.*

If $\{L_i \subseteq A_i^*, \; i \in Z_n\}$ are mutually normal then

$$\sup \mathrm{N}(\|_{i=1}^{n} K_i, \|_{i=1}^{n} L_i, P) = \|_{i=1}^{n} \sup \mathrm{N}(K_i, L_i, P_i^{\mathrm{loc}}). \tag{21}$$

Conversely, if for fixed local plant languages $\{L_i, \; i \in Z_n\}$ Eq. (21) holds for any $\{K_i \subseteq L_i, i \in \mathbb{Z}_n\}$, then

$$P_i(L) \text{ are normal with respect to } L_i \text{ and } P_i^{\mathrm{loc}}, \forall i \in \mathbb{Z}_n. \tag{22}$$

The proof of the above theorem for one direction depends only on the assumption that the local plants agree on the observability of their common events. Hence the following corollary is obtained.

Corollary 5.12. *If the local plants agree on the observability of their common events then we have*

$$\sup \mathrm{N}(\|_{i=1}^{n} K_i, \|_{i=1}^{n} L_i, P) \supseteq \|_{i=1}^{n} \sup \mathrm{N}(K_i, L_i, P_i^{\mathrm{loc}}). \tag{23}$$

Theorem 5.9 is useful for the computation of (global) supremal controllable and normal sublanguages of large distributed plants. If the conditions of the theorem are satisfied, then it is sufficient to compute local supremal controllable and normal sublanguages and to synchronize these.

The interest of this theorem should be clear: under the conditions which are stated it is possible to do the optimal (least restrictive) control synthesis with partial observations locally, and this represents an exponential savings on the computational complexity and makes in fact the optimal control synthesis of large distributed plants feasible.

In the sequel it will be shown that mutual normality cannot be satisfied in the so-called shuffle case (the case, where event sets are pairwise disjoint). However this is a much simpler case where mutual normality is not needed. In fact, we have

Theorem 5.13. *Modular control synthesis equals global control synthesis in the shuffle case of a distributed DES. Assume that the local alphabets are pairwise disjoint, i.e. $A_i \cap A_j = \emptyset$ for any $i, j \in \mathbb{Z}_n$ with $i \neq j$. Then for any decomposable specification $K \subseteq L$*

$$\|_{i=1}^n \sup \mathrm{N}(K_i, L_i, P_i^{\mathrm{loc}}) = \sup \mathrm{N}(\|_{i=1}^n K_i, \|_{i=1}^n L_i, P). \tag{24}$$

Proof. In the shuffle case mutual controllability is trivially satisfied and we show that mutual normality is not needed. The proof relies on Algorithm 1 specialized to the case $A_c = A$, i.e. $A_u = \emptyset$. We work with the behaviors (languages) generated by the automata representations of the globally and locally supremal normal sublanguages resulting from their computations according to Algorithm 1. The notation is as follows: let S representing K and T representing L are such that S is a subautomaton of T and S is a state-partition automaton. The transition functions of S and T are denoted by $\rightarrow$ and $\rightarrow_1$, respectively. Algorithm 1 yields partial automaton $\tilde{S} = \langle \tilde{o}, \tilde{t} \rangle$ with $\tilde{t}$ denoted by $\rightarrow_{'}$ and its behavior by $\tilde{l} : \tilde{S} \rightarrow \mathcal{L}$.
Similarly, for $i \in \mathbb{Z}_n$, S_i and T_i representing K_i and L_i, respectively, are such that S_i is a subautomaton of T_i and S_i is a state-partition automaton. The transition functions of S_i and T_i are denoted by $\rightarrow_{1i}$ and $\rightarrow_i$, respectively. Construction of Algorithm 1 yields partial automaton $\tilde{S}_i = (\tilde{S}_i, \langle \tilde{o}_i, \tilde{t}_i \rangle)$ with $\tilde{t}_i$ denoted by $\rightarrow_{i'}$ and its behavior by $\tilde{l}_i : \tilde{S} \rightarrow \mathcal{L}$. The (common) initial state of S and T is denoted by s_0 and for $i \in \mathbb{Z}_n$ the (common) initial states of S_i and T_i are denoted by s_0^i. The transition function of S_i and S is denoted by $\rightarrow_1$ and the transition function of T_i and T is denoted by $\rightarrow$. Therefore, $\tilde{l}(s_0) = \sup \mathrm{N}(K, L, P)$ and for any $i \in \mathbb{Z}_n$: $\tilde{l}_i(s_0^i) = \sup \mathrm{N}(K_i, L_i, P_i^{\mathrm{loc}})$.

It is sufficient to show that

$$R = \{\langle [\tilde{l}(s_0)]_v, [\|_{i=1}^n \tilde{l}_i(s_0^i)]_v \rangle \mid v \in (\tilde{l}(s_0))^2\}$$

is a bisimulation relation, from which the claim of the theorem follows by coinduction. From Corollary 5.12 it follows that only one inclusion is to be shown.

Let $[\tilde{l}(s_0)]_v \xrightarrow{a}$, i.e. condition (*) of Algorithm 1 is satisfied. Note that $[\|_{i=1}^n \tilde{l}_i(s_0^i)]_v = \|_{i=1}^n [\tilde{l}_i(s_0^i)]_{v_i}$, where $v_i := P_i(v)$. We show that $\forall i \in \mathbb{Z}_n : [\tilde{l}_i(s_0^i)]_{v_i} \xrightarrow{a}$, i.e. $l_i(s_0^i)_{v_i} \xrightarrow{a}$. According to Algorithm 1 applied to S_i and T_i we must show that condition (*) holds. Let $a \in A$. Then there exists one and only one $i \in \mathbb{Z}_n$ such that $a \in A_i$. We have two possibilities: either $a \in A_{uo,i}$ or $a \in A_{o,i}$. We first take $a \in A_{uo,i} \subseteq A_{uo}$. According to Algorithm 1 it is sufficient to show that $\forall u_i \in A_{uo,i}^* : (s_0^i)_{v_i a} \xrightarrow{u_i}_i \Rightarrow (s_0^i)_{v_i a} \xrightarrow{u_i}_{1i}$. In the shuffle case for $a \in A_{uo,i} \subseteq A_{uo}$ we have $\forall j \neq i : P_j(a) = \varepsilon$. Let $u_i \in A_{uo,i}^*$: $(s_0^i)_{v_i a} \xrightarrow{u_i}_i$. Hence, $v_i a u_i \in L_i$. We know that condition (*) holds for $\tilde{S}$, i.e. $\forall u \in A_{uo}^*$: $(s_0)_{va} \xrightarrow{u} \Rightarrow (s_0)_{va} \xrightarrow{u}_1$. In order to use this assumption it must be shown that $(s_0)_{va} \xrightarrow{u}$ for a $u \in A_{uo}^*$, i.e. $vau \in L = \|_{i=1}^n L_i$. Let us take $u := u_i$. Then using once more the property of the shuffle case $P_i(u) = u_i$, while $\forall j \neq i : P_j(u) = \varepsilon$. We already know that $vau \in P_i^{-1} L_i$, because $v_i a u_i = P_i(vau) \in L_i$. For any $j \neq i$ we get trivially : $P_j(vau) = v_j \in L_j$, because $v \in L$. Therefore $vau \in L$ and $(s_0)_{va} \xrightarrow{u}$. Thus, $(s_0)_{va} \xrightarrow{u}_1$, which means that $vau \in K$, i.e. $v_i a u_i = P_i(vau) \in K_i$. Equivalently, $(s_0^1)_{v_i a} \xrightarrow{u_i}_{1i}$, which was to be shown.

Now let $a \in A_{o,i} \subseteq A_o$ then we know that $\forall s' \approx_{\mathrm{Aux}(S)} (s_0)_v : s' \xrightarrow{a} \Rightarrow s' \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo}^*$: $s'_a \xrightarrow{u} \Rightarrow s'_a \xrightarrow{u}_1$. It must be shown that $(s_0^i)_{v_i} \xrightarrow{a}_{'}$, i.e. $\forall q^i \approx_{\mathrm{Aux}(S_i)} (s_0^i)_{v_i} : q^i \xrightarrow{a}_i \Rightarrow q^i \xrightarrow{a}_{1i}$, in which case also $\forall u_i \in A_{uo,i}^*$: $q_a^i \xrightarrow{u_i}_i \Rightarrow q_a^i \xrightarrow{u_i}_{1i}$. Let $q^i \approx_{\mathrm{Aux}(S_i)} (s_0^i)_v : q^i \xrightarrow{a}_i$. Since S_i is a state-partition automaton, there exists $v'_i \in A_i^*$ such that $P_i^{\mathrm{loc}}(v'_i) = P_i^{\mathrm{loc}}(v_i)$ and $q^i = (s_0^i)_{v'_i}$. Since $q^i \xrightarrow{a}_i$ we have $v'_i a \in K_i \subseteq L_i$. Then $v'a \in P_1^{-1}(v'_1 a)$. Therefore $v'a \in P_i^{-1}(L_i)$. We show that $v'a \in P_j^{-1}(L_j)$ for all $j \neq i$. Indeed, $P_j(v'a) = v'_j \in L_j$, because $v' \in L$. Therefore $v'a \in P_j^{-1}(L_j)$. Thus, $v'a \in L = \cap_{i=1}^n P_i^{-1}(L_i)$. Since $P(v') = P(v)$, we have $(s_0)_{v'} \approx_{\mathrm{Aux}(S)} (s_0)_v$. From $(s_0)_v \xrightarrow{a}_{'}$ and condition (*) of Algorithm 1, it follows that $(s_0)_{v'} \xrightarrow{a} \Rightarrow (s_0)_{v'} \xrightarrow{a}_1$, and also $\forall u \in A_{uo}^*$:

$(s_0)_{v'a} \xrightarrow{u} \Rightarrow \ (s_0)_{v'a} \xrightarrow{u}_1$. But this implies that $(s_0^i)_{v_i'} = q^i \xrightarrow{a}_{1i}$, because $v'a \in K = \|_{i=1}^n K_i$ implies that $v_i'a = P_i(v'a) \in K_i$. We show also that $\forall u_i \in A_{uo,i}^*$: $q_a^i \xrightarrow{u_i}_i \Rightarrow q_a^i \xrightarrow{u_i}_{1i}$. Indeed, $q_a^i \xrightarrow{u_i}$ means $v_i'au_1 \in L_i$. Similarly as for $a \in A_{uo}$, by considering $u = u_i \in A_{uo}^*$ we obtain using shuffle property that $v'au \in L$, i.e. $(s_0)_{v'a} \xrightarrow{u}$ whence $(s_0)_{v'a} \xrightarrow{u}_1$. But this means that $v'au \in K$, i.e. $v_i'au_i = P_i(v'au) \in K_i$, or equivalently $q_a^i \xrightarrow{u_i}_{1i}$. $\square$

Example and verification of sufficient conditions

The purpose of this section is mainly to illustrate our results with an example. Before starting with concrete examples we consider several extreme cases of distributed DES. First of all, if all event alphabets are disjoint, the so-called shuffle case, we notice that $P_i(P_j)^{-1}(L_j) = A_i^*$ for any $L_j \subseteq A_j^*$. This means that the condition of mutual normality cannot be satisfied. In view of Theorem 5.13 this is not a problem.

On the other hand, it is obvious from the definition of mutual normality that in the case of full local observations (all P_i^{loc}'s become identity mappings), mutual normality is trivially satisfied. Another extreme case occurs when all subsystems have the same event alphabets. Then all the P_i's are identity mappings, i.e. the mutual normality becomes usual normality between two languages in a slightly more general sense (the assumption is lifted that one of the languages is a sublanguage of the other). This might justify why we call our condition mutual normality, it is a symmetric notion of normality.

We show an example of a plant composed of two modules, where the commutativity between the supremal normal sublanguages and parallel product does not hold. Therefore mutual normality does not hold either.

Example 5.14. Let $A = \{a, a_1, a_2, \tau, \tau_1, \tau_2\}$, $A_1 = \{a_1, \tau_1, a, \tau\}$, $A_2 = \{a_2, \tau_2, a, \tau\}$, $A_o = \{a_1, a_2, a\}$, $A_{o,1} = \{a_1, a\}$, and $A_{o,2} = \{a_2, a\}$. Consider the following plant languages and specification sublanguages (the marked languages are not considered):

$$\begin{tikzcd}
& K_1 = L_1 \arrow[dl, "\tau"'] \arrow[dr, "\tau_1"] & \\
(K_1)_\tau \arrow[d, "a"'] & & (K_1)_{\tau_1} \arrow[d, "a_1"'] \\
(K_1)_{\tau a} & & (K_1)_{\tau_1 a_1}
\end{tikzcd}
\qquad
\begin{tikzcd}
& K_2 \arrow[dl, "\tau"'] \arrow[dr, "\tau_2"] & \\
(K_2)_\tau \arrow[d, "a"'] & & (K_2)_{\tau_2} \\
(K_2)_{\tau a} & &
\end{tikzcd}
\qquad
\begin{tikzcd}
& L_2 \arrow[dl, "\tau"'] \arrow[dr, "\tau_2"] & \\
(L_2)_\tau \arrow[d, "a"'] & & (L_2)_{\tau_2} \arrow[d, "a"'] \\
(L_2)_{\tau a} & & (L_2)_{\tau_2 a}
\end{tikzcd}$$

We use the notation $U_1 = \sup \mathrm{N}(K_1, L_1, P_1^{\mathrm{loc}})$, $U_2 = \sup \mathrm{N}(K_2, L_2, P_2^{\mathrm{loc}})$, $U = \sup \mathrm{N}(K_1, L_1, P_1^{\mathrm{loc}}) \parallel \sup \mathrm{N}(K_2, L_2, P_2^{\mathrm{loc}})$, and $V = \sup \mathrm{N}(K_1 \parallel K_2, L_1 \parallel L_2, P)$. We have trivially that $U_1 = K_1 = L_1$. It is easy to see that $U_2 = \sup \mathrm{N}(K_2, L_2, P_2^{\mathrm{loc}}) = \{\varepsilon, \tau, \tau_2\}$. Computing the parallel products $K = K_1 \parallel K_2$ and $L = L_1 \parallel L_2$ yields $K = L$, i.e. we obtain trivially $K = L = V$ as is shown in the diagram below, where $U = U_1 \parallel U_2$ is also computed:

$$\begin{tikzcd}
& U_2 \arrow[dl, "\tau"'] \arrow[dr, "\tau_2"] & \\
(U_2)_\tau & & (U_2)_{\tau_2}
\end{tikzcd}
\qquad
\begin{tikzcd}
& U \arrow[dl, "\tau"'] \arrow[d, "\tau_1"] \arrow[dr, "\tau_2"] & & \\
U_\tau & U_{\tau_1} \arrow[d, "a_1"'] \arrow[dr, "\tau_2"] & U_{\tau_2} \arrow[dr, "\tau_1"] & \\
& U_{\tau_1 a_1} \arrow[d, "\tau_2"'] & U_{\tau_1 \tau_2} \arrow[d, "a_1"'] & U_{\tau_2 \tau_1} \arrow[d, "a_1"'] \\
& U_{\tau_1 a_1 \tau_2} & U_{\tau_1 \tau_2 a_1} & U_{\tau_2 \tau_1 a_1}
\end{tikzcd}
\qquad
\begin{tikzcd}
& K = L = V \arrow[dl, "\tau"'] \arrow[d, "\tau_1"] \arrow[dr, "\tau_2"] & & \\
K_\tau \arrow[d, "a"'] & K_{\tau_1} \arrow[d, "a_1"'] \arrow[dr, "\tau_2"] & K_{\tau_2} \arrow[dr, "\tau_1"] & \\
K_{\tau a} & K_{\tau_1 a_1} \arrow[d, "\tau_2"'] & K_{\tau_1 \tau_2} \arrow[d, "a_1"'] & K_{\tau_2 \tau_1} \arrow[d, "a_1"'] \\
& K_{\tau_1 a_1 \tau_2} & K_{\tau_1 \tau_2 a_1} & K_{\tau_2 \tau_1 a_1}
\end{tikzcd}$$

Thus, $U \neq V$, because $U_\tau \not\xrightarrow{a}$, while $V_\tau \xrightarrow{a}$. Therefore we only have the strict inclusion $U \subset V$ and the commutativity studied in this paper does not hold for this example. According to Theorem 5.11 mutual normality cannot hold.

Indeed, we have

$$(P_1^{\mathrm{loc}})^{-1} P_1^{\mathrm{loc}}(L_1) \cap P_1(P_2)^{-1}(L_2) = \tau_1^*(\tau\tau_1^* a_1 + a_1\tau_1^*\tau)\tau_1^*,$$

but we have e.g. $(\tau_1)^n \notin L_1$ for $n \geq 2$.

Antipermissive control policy

The standard permissive control law is useful for safety control problems only if the specification is observable, otherwise it yields infimal observable and controllable superlanguages of K. If K represents safety specifications then these are violated if permissive control policy is applied.

There exists a dual control policy for DES with partial observations, called *antipermissive*. For the supervisor to enable event $a \in A$ it is necessary that all indistinguishable events corresponding to this trace can be prolongated within K (in permissive control policy it is sufficient that one of those strings can be prolongated within K). The interest of the antipermissive control policy concerns the safety control objective and the fact that the synthesized languages are observable languages which are in general larger than supremal normal sublanguages.

Denote by S_A the antipermissive control policy for a partial automaton S: $S_A : P(L(G)) \to \Gamma_c$, where Γ_c is the class of enabled events, also called control patterns (i.e. supersets of the event subset A_u that are always enabled). Algebraically, the antipermissive control policy is defined as follows:

$$S_A(s) = A_u \cup \left\{ \begin{matrix} a \in A_c : \ \forall s' \in \mathrm{prefix}(K) \cap P^{-1}P(s) \\ (s'a \in L \Rightarrow s'a \in \mathrm{prefix}(K)) \end{matrix} \right\}. \tag{25}$$

Similarly as for the permissive control policy, the supervisor marks all states that have been marked in the plant and that 'survive' under supervision.

We have formulated in [10] a single-step algorithm for computation of closed-loop languages with respect to the antipermissive control policy. This algorithm has been used in [6] for deriving sufficient conditions under which these languages are preserved by modular (local) control synthesis.

Denote for all $i \in \mathbb{Z}_n$ by $\mathrm{AP}(K_i, L_i, P_i^{\mathrm{loc}})$ the closed-loop language corresponding to the local antipermissive control synthesis (with local projection P_i^{loc}, local DES L_i, and local specification K_i). Similarly, $\mathrm{AP}(\|_{i=1}^n K_i, \|_{i=1}^n L_i, P)$ stands for closed-loop language corresponding to the global antipermissive control synthesis.

In Theorem 5.16 a condition similar to mutual controllability [25] is used. By analogy it is called mutual observability.

Definition 5.15. Consider a distributed DES. The local plant languages $\{L_i \subseteq A_i^*, \ i \in \mathbb{Z}_n\}$ are called *mutually observable* if

$$\begin{aligned} &\forall i, j \in \mathbb{Z}_n, \ i \neq j, \ \forall s, s' \in L_i \text{ and } \forall a \in A_{ic} : \\ &(sa \in P_i(P_j)^{-1}(L_j) \text{ and } P_i^{\mathrm{loc}}(s) = P_i^{\mathrm{loc}}(s') \text{ and } s'a \in L_i) \\ &\Rightarrow sa \in L_i. \end{aligned} \tag{26}$$

Now we extend the main result of [6] where only sufficient conditions were presented.

Theorem 5.16. *Modular control synthesis equals global control synthesis for the closed-loop language in the case of local specifications using an antipermissive control policy. Assume that the modular plants agree on the observability of their common events.*

If the local plant languages $\{L_i \subseteq A_i^*, \ i \in Z_n\}$ are mutually observable then for any decomposable specification $K \subseteq L$

$$\mathrm{AP}(\|_{i=1}^n K_i, \|_{i=1}^n L_i, P) = \|_{i=1}^n \mathrm{AP}(K_i, L_i, P_i^{\mathrm{loc}}). \tag{27}$$

Conversely, if for fixed local plant languages $\{L_i \ \ i \in Z_n\}$ Eq. (27) holds for any $\{K_i \subseteq L_i, i \in \mathbb{Z}_n\}$, then $\{P_i(L), \ i \in \mathbb{Z}_n\}$ are observable with respect to L_i and $A_{o,i}$.

Proof. The sufficiency part can be found in [6] by coinduction. In view of our previous results it is straightforward to show that a necessary structural condition for local antipermissive control policy to equal global antipermissive control policy is that $\mathrm{AP}(P_i(L), L_i, P_i^{\mathrm{loc}}) = P_i(L)$, which means that for all $i \in \mathbb{Z}_n$, $P_i(L)$ is observable with respect to L_i and $A_{i,o}$. □

6. Modular supervisory control with an indecomposable specification

In many engineering problems the specification is only defined globally and is not decomposable unlike the plant language. An example is the specification for a communication protocol of a wireless network. In this section the case of general specification languages that are neither necessarily decomposable nor contained in the global plant language is studied. Necessary and sufficient conditions are found with respect to which handling of the global plant is avoided for the computation of supremal controllable sublanguages of (global) indecomposable specification languages.

Consider a modular discrete-event system and assume that the local plants agree on the controllability of their common events. Denote the global plant and the specification languages by L and K, respectively. In our setting, L is decomposable into local plant languages: $L = L_1 \parallel \cdots \parallel L_n$ (note that the L_i may have different alphabets). In most of the works on this topic K is similarly decomposable into local specification languages and $K \subseteq L$. The general case is when this condition is not satisfied and, moreover, K may not be included in L. This case has been studied in [3], where the assumption that all shared events are controllable is used. A condition on K called G-observability was needed for local synthesis of the supremal controllable sublanguage.

Instead of local specifications, the languages $K_i := K \cap P_i^{-1}(L_i)$ are considered. These will play the role of local components of specification languages, although their alphabet is the global alphabet A. They can be considered as local over-approximations of $K \cap L$, because clearly $K \cap L = \cap_{i=1}^n K_i$.

Definition 6.1. Consider a modular DES. The modular plant languages $\{L_i,\ i \in Z_n\}$ are called *globally mutually controllable* if

$$P_j^{-1}(L_j)(A_{ju}) \cap P_i^{-1}(L_i) \subseteq P_j^{-1}(L_j), \forall i, j \in \mathbb{Z}_\text{n}, i \neq j. \tag{28}$$

Proposition 6.2. *Consider a modular DES. Global mutual controllability (GMC) is equivalent to the following property:*

$$P_j^{-1}(L_j)A_u \cap P_i^{-1}(L_i) \subseteq P_j^{-1}(L_j), \forall i, j \in \mathbb{Z}_\text{n}, i \neq j. \tag{29}$$

Proof. Note that the only difference is that A_{ju} in GMC is replaced by A_u. Therefore the new property is clearly stronger then GMC. Maybe surprisingly the converse implication is satisfied as well. Let GMC hold true, $s \in P_j^{-1}(L_j)$, $u \in A_u$, and $su \in P_i^{-1}(L_i)$. Then we have two cases: either $u \in A_j$ or $u \notin A_j$. The former case entails that $u \in A_{ju}$ due to the shared event controllability status assumption. Using GMC we conclude $su \in P_j^{-1}(L_j)$. In the latter case we notice that $P_j(u) = \varepsilon$, i.e. $P_j(su) = P_j(s)$, which means that $P_j(su) \in L_j$, i.e. $su \in P_j^{-1}(L_j)$. □

Definition 6.3. Consider a modular DES. The modular plant languages $\{L_i, i \in Z_n\}$ are called *modularly controllable* if

$$LA_u \cap P_i^{-1}(L_i) \subseteq L, \forall i \in \mathbb{Z}_\text{n}. \tag{30}$$

Modular controllability is in general weaker than global mutual controllability and it will play the role of a necessary condition.

Proposition 6.4. *Consider a modular DES. Global mutual controllability (GMC) implies modular controllability.*

Proof. Let for any $i \neq j \in \text{Z}_n$:

$$P_j^{-1}(L_j)A_u \cap P_i^{-1}(L_i) \subseteq P_j^{-1}(L_j).$$

Since for $j = i$ the inclusion is trivially true, we have the inclusion for any $i, j \in \text{Z}_n$. Then, by taking intersection for j over Z_n we obtain:

$$\bigcap_{j=1}^{n}[P_j^{-1}(L_j)A_u \cap P_i^{-1}(L_i)] \subseteq \bigcap_{j=1}^{n} P_j^{-1}(L_j) = L.$$

Since $[\bigcap_{j=1}^n P_j^{-1}(L_j)]A_u \subseteq \bigcap_{j=1}^n [P_j^{-1}(L_j)A_u]$ we obtain finally,

$$LA_u \cap P_i^{-1}(L_i) = \left[\bigcap_{j=1}^{n} P_j^{-1}(L_j)\right] A_u \cap P_i^{-1}(L_i) \subseteq L,$$

i.e. modular controllability (MC). □

As for the opposite inclusion, it turns out that it holds only for $n = 2$. We have

Proposition 6.5. *Consider a modular DES. For $n = 2$ global mutual controllability (GMC) is equivalent to modular controllability.*

Proof. According to Proposition 6.4 it is sufficient to show that for $n = 2$ modular controllability implies GMC. Let MC holds. We show that

$$P_j^{-1}(L_j)A_u \cap P_i^{-1}(L_i) \subseteq P_j^{-1}(L_j).$$

Let $s \in P_j^{-1}(L_j)$, $u \in A_u$, and $su \in P_i^{-1}(L_i)$. Then we have $s \in P_i^{-1}(L_i)$ as well, because $P_i^{-1}(L_i)$ is prefix-closed. Since we have two modules and $i \neq j$ we obtain $s \in L$. Now using MC

$$su \in LA_u \cap P_i^{-1}(L_i) \subseteq L \subseteq P_j^{-1}(L_j),$$

whence $su \in P_j^{-1}(L_j)$ and GMC holds. □

For a number of modules greater than or equal to 3 however, GMC and MC differ, because we cannot deduce $s \in L$ from $s \in P_j^{-1}(L_j)$, $u \in A_u$, and $su \in P_i^{-1}(L_i)$. This is illustrated through the following simple example:

Example 6.6.

$$\begin{aligned}
L_1 &= \{\text{prefix}(u_{13}^*(uu_{13}^* + au_{13}))\},\ L_2 = \{\text{prefix}(au + u)\},\\
L_3 &= \{\text{prefix}(u_{13}^* u u_{13}^*)\}\\
A_1 &= \{a, u, u_{13}\} = A_3,\ A_2 = \{a, u\},\ A_{1,c} = \{a\} = A_{2,c} = A_{3,c}.
\end{aligned}$$

The modules agree on controllability of common events because

$$A_i \cap A_{j,u} = \{u\} = A_{j,u} \cap A_i,\ i = 2,\ j \in \{1, 3\},$$
$$A_1 \cap A_{3,u} = \{u, u_{13}\} = A_{1,u} \cap A_3.$$

Since $L = L_3$, it is easy to see that modular controllability holds for this example. However, L_1 and L_2 are not GMC, because $au_{13}u_{13} \in P_1^{-1}(L_1)A_{1u} \cap P_2^{-1}(L_2) \setminus P_1^{-1}(L_1)$.

The next theorem provides novel necessary and sufficient conditions for modular control synthesis to equal global control synthesis. In [12] it was shown only that global mutual controllability is a sufficient condition.

Theorem 6.7. *Modular control synthesis equals global control synthesis for the supremal controllable sublanguage in the case of complete observations and of indecomposable specifications. Consider a modular discrete-event system. Assume that the local plants agree on the controllability of their common events.*

If the local plants $\{L_i,\ i \in Z_n\}$ are modularly controllable then for any indecomposable specification $K \subseteq A^*$,

$$\sup \mathrm{C}(K \cap L, L, A_u) = \bigcap_{i=1}^{n} \sup \mathrm{C}(K_i, P_i^{-1}(L_i), A_u). \tag{31}$$

Conversely, if for a given modular plant equality (31) holds for any global specification K then the local plant languages, $\{L_i,\ i \in Z_n\}$ are modularly controllable.

Proof. For sufficiency the coinductive proof principle will be used, i.e. it is sufficient to show that under the conditions listed above the following relation (cf. coinductive definition and notation for sup C!)

$$R = \left\{\left\langle (K \cap L)/_{A_u}^{SC} L, \bigcap_{i=1}^{n} [K_i/_{A_u}^{SC} P_i^{-1}(L_i)] \right\rangle;\ K, L \in \mathcal{L}\right\}$$

is a bisimulation relation, from which the equality follows by coinduction. In order to show that R is a bisimulation of partial languages we must show the points (i)–(iii) below (see Definition 3.4 of Section 3).

(i) This item is obvious from the coinductive definition of supremal controllable sublanguages.

(ii) If for $a \in A$: $\bigcap_{i=1}^n [K_i/^{SC}_{A_u} P_i^{-1}(L_i)] \xrightarrow{a}$ then $\forall i \in \mathbb{Z}_n$: $[K_i/^{SC}_{A_u} P_i^{-1}(L_i)] \xrightarrow{a}$. Thus, we have for $a \in A$ that $\forall i \in \mathbb{Z}_n : K_i \xrightarrow{a}$, $P_i^{-1}(L_i) \xrightarrow{a}$ and $\forall u \in A_u^* :\ P_i^{-1}(L_i)_a \xrightarrow{u} \Rightarrow\ (K_i)_a \xrightarrow{u}$.

We must show that $(K \cap L)/^{SC}_{A_u} L \xrightarrow{a}$, which according to the coinductive definition of the supremal controllable sublanguage means that $(K \cap L) \xrightarrow{a}$, $L \xrightarrow{a}$, and $\forall u \in A_u^* :\ L_a \xrightarrow{u} \Rightarrow\ (K \cap L)_a \xrightarrow{u}$. First of all, $(K \cap L) \xrightarrow{a}$ immediately follows from $K_i \xrightarrow{a}$, because $K \cap L = K \cap \bigcap_{i=1}^n P_i^{-1}(L_i) = \bigcap_{i=1}^n [K \cap P_i^{-1}(L_i)] = \bigcap_{i=1}^n K_i$. Obviously, $(K \cap L) \xrightarrow{a}$ implies $L \xrightarrow{a}$. Now let $u \in A_u^*$ such that $L_a \xrightarrow{u}$. Then $\forall i \in \mathbb{Z}_n :\ P_i^{-1}(L_i)_a \xrightarrow{u}$, whence $\forall i \in \mathbb{Z}_n :\ (K_i)_a \xrightarrow{u}$. It follows that $(K \cap L) \xrightarrow{a}$.

(iii) Let $(K \cap L)/^{SC}_{A_u} L \xrightarrow{a}$ for $a \in A$. It follows from the coinductive definition of the supremal controllable sublanguage that $(K \cap L) \xrightarrow{a}$, $L \xrightarrow{a}$, and $\forall u \in A_u^* :\ L_a \xrightarrow{u} \Rightarrow\ (K \cap L)_a \xrightarrow{u}$. We need to show that $\cap_{i=1}^n [K_i/^{SC}_{A_u} P_i^{-1}(L_i)] \xrightarrow{a}$, i.e. that $\forall i \in \mathbb{Z}_n : [K_i/^{SC}_{A_u} P_i^{-1}(L_i)] \xrightarrow{a}$. First of all we notice that $\forall i \in \mathbb{Z}_n :\ K_i \xrightarrow{a}$, because as we have seen in (ii) above, $\bigcap_{i=1}^n K_i = K \cap L \xrightarrow{a}$. In the very same way, $\forall i \in \mathbb{Z}_n :\ P_i^{-1}(L_i) \xrightarrow{a}$ follows from $L \xrightarrow{a}$. Now let $P_i^{-1}(L_i)_a \xrightarrow{u}$ for some $u \in A_u^*$, i.e. $au \in P_i^{-1}(L_i)$. Due to our assumption of modular controllability $au \in LA_u \cap P_i^{-1}(L_i) \subseteq L$, hence $au \in L$. According to our assumption we deduce from $L_a \xrightarrow{u}$ that $(K \cap L)_a \xrightarrow{u}$. Therefore for any $i \in \mathbb{Z}_n$ $(K_i)_a \xrightarrow{u}$, which was to be shown.

Now we will prove the second part of the theorem. The necessity of modular controllability is easy to show: Let $L_i,\ i \in \mathbb{Z}_n$ be given and let Eq. (31) hold for any $K \subseteq A^*$. Then in particular (the more difficult) inclusion

$$\sup C(K \cap L, L, A_u) \subseteq \bigcap_{i=1}^n \sup C(K_i, P_i^{-1}(L_i), A_u)$$

holds for any $K \subseteq A^*$. This is equivalent to

$$\sup C(K \cap L, L, A_u) \subseteq \sup C(K_i, P_i^{-1}(L_i), A_u), \forall K \subseteq A^* \text{ and } i \in \mathbb{Z}_n.$$

Then the inclusion holds in particular for $K = L$, which entails $K_i = L$ for any $i \in \mathbb{Z}_n$. Hence, $L = \sup C(K \cap L, L, A_u) \subseteq \sup C(L, P_i^{-1}(L_i), A_u)$. Since $\sup C(L, P_i^{-1}(L_i), A_u) \subseteq L$ by definition of the $\sup C$ operator, we obtain $L = \sup C(L, P_i^{-1}(L_i), A_u)$. But this means that L is controllable with respect to $P_i^{-1}(L_i)$ and A_u for any $i \in \mathbb{Z}_n$, which is the modular controllability condition. □

Note that as a consequence of Theorem 6.7 global mutual controllability is not a necessary condition, because modular controllability is both necessary and sufficient structural condition and although for $n = 2$ these two conditions coincide, for $n \geq 3$ global mutual controllability is in general stronger than modular controllability. The computational time complexity for both global and modular control synthesis in this case is discussed in the Sections 8.2 and 8.3.

Global mutual controllability is quite a strong condition. We show a simple example, where global mutual controllability holds. Consequently, according to Theorem 6.7 modular and global computation of supremal controllable sublanguage coincide for all specifications $K \subseteq A^*$.

Example 6.8.

$$\begin{aligned} L_1 &= \{\varepsilon, a, au, a_1, a_1u\}, L_2 = \{\varepsilon, a, au, a_2, a_2u\}, \\ A_1 &= \{a, u, a_1\}, A_2 = \{a, u, a_2\}, A_{1,c} = \{a, a_1\} A_{2,c} = \{a, a_2\}. \end{aligned}$$

Modules agree on controllability of common events,

$A_1 \cap A_{2,u} = \{u\} = A_{1,u} \cap A_2$.

It is easy to see that global mutual controllability holds for this example. Therefore we have

$$\begin{aligned} &\sup C(K \cap P_1^{-1}(L_1), P_1^{-1}(L_1), A_u) \cap \sup C(K \cap P_2^{-1}(L_2), P_2^{-1}(L_2), A_u) \\ &\quad = \sup C(K \cap L, L, A_u), \forall K \subseteq A^*, \end{aligned}$$

i.e. modular and global computation of supremal controllable sublanguage coincide for all specifications $K \subseteq A^*$.

7. Distributed supervisory control with an indecomposable specification

In this section the case is studied of control of a distributed discrete-event system, thus for which one or more of the local plants has only partial observations, and of an indecomposable specification. First a structural condition called global mutual normality is introduced. It is similar to mutual normality in the case of decomposable specification, but it concerns $P_i^{-1}(L_i)$ instead of L_i itself.

Definition 7.1. Consider a distributed DES. The modular plant languages $\{L_i \subseteq A_i^*,\ i \in \mathbb{Z}_\mathrm{n}\}$ are called *globally mutually normal* if

$$(P^{-1}PP_j^{-1})(L_j) \cap P_i^{-1}L_i \subseteq P_j^{-1}L_j, \forall i, j \in \mathbb{Z}_\mathrm{n},\ i \neq j. \tag{32}$$

In the next definition the concept needed for a necessary and sufficient condition is introduced.

Definition 7.2. Consider a distributed DES. Modular plant languages $\{L_i,\ i \in Z_n\}$ are called *modularly normal* if L is $(P_i^{-1}(L_i), P)$-normal; or, equivalently,

$$P^{-1}P(L) \cap P_i^{-1}(L_i) \subseteq L, \forall i \in \mathbb{Z}_\mathrm{n}. \tag{33}$$

Modular normality is in general weaker than global mutual normality (GMN).

Proposition 7.3. *Consider a distributed DES. Global mutual normality (GMN) implies modular normality (MN).*

Proof. Let GMN holds, or, equivalently,

$$(P^{-1}PP_j^{-1})(L_j) \cap P_i^{-1}L_i \subseteq P_j^{-1}L_j, \forall i, j \in \mathbb{Z}_\mathrm{n},\ i \neq j.$$

Since for $i = j$ the inclusion becomes trivial, we may assume that the inclusion is satisfied for any $i, j \in Z_n$. We obtain:

$$\begin{aligned} P^{-1}P(L) \cap P_i^{-1}(L_i) &= P^{-1}P\left(\bigcap_{j=1}^{n} P_j^{-1}(L_j)\right) \cap P_i^{-1}(L_i) \\ &\subseteq \bigcap_{j=1}^{n}(P^{-1}PP_j^{-1})(L_j) \cap P_i^{-1}(L_i) \subseteq \bigcap_{j=1}^{n} P_j^{-1}(L_j) = L, \end{aligned}$$

where the last inclusion follows from intersecting both sides of the first inclusion (GMN) for j ranging over $\mathbb{Z}_\mathrm{n}$. Thus MN holds. □

7.1. Modular computation of supremal controllable and normal sublanguages

Theorem 7.4. *Modular control synthesis equals global control synthesis for the supremal controllable and normal sublanguage in the case of a distributed DES and of an indecomposable specification. Consider a distributed DES. Assume that the local plants* (1) *agree on the controllability of their common events and* (2) *agree on the observability of their common events.*

If the local plant languages $\{L_i, i \in Z_n\}$ are modularly controllable and modularly normal, then for any indecomposable specification $K \subseteq A^*$

$$\sup \mathrm{CN}(K \cap L, L, P, A_u) = \bigcap_{i=1}^{n} \sup \mathrm{CN}(K_i, P_i^{-1}(L_i), P, A_u). \tag{34}$$

Conversely, if for a given modular plant equality (34) *holds for any global specification K then the local plant languages $\{L_i,\ i \in Z_n\}$ are modularly controllable and modularly normal.*

Proof. The sufficiency part of the proof relies on Algorithm 1 for computation of sup CN and the coinduction proof principle. Algorithm 1 is used for the computation of $\sup \mathrm{CN}(K \cap L, L, P)$ as well as for computation of $\sup \mathrm{CN}(K_i, P_i^{-1}(L_i), P)$ with the first two parameters represented by partial automata S_i and T_i defined below. Let

S representing $K \cap L$ and T representing L be the same as in Algorithm 1. Algorithm 1 yields the partial automaton $\tilde{S} = (\tilde{S}, \langle \tilde{o}, \tilde{t} \rangle)$, subautomaton of S, that represents $\sup \mathrm{N}(K \cap L, L, P)$ according to Theorem 5.7.

Similarly, let partial automata $S_i = (S_i, \langle o_{1i}, t_{1i} \rangle)$ and $T_i = (T_i, \langle o_i, t_i \rangle)$ representing K_i and $P_i^{-1}(L_i)$, $i \in \mathbb{Z}_n$, respectively, be such that for all $i \in \mathbb{Z}_n$: S_i is a subautomaton of T_i, and S_i is a state-partition automaton. The common initial state of these automata is denoted by s_0^i and the transition functions t_{1i} and t_i are denoted by $\rightarrow_{1i}$ and $\rightarrow_i$, respectively. Denote by $\mathrm{Aux}(S_i)$ the observation indistinguishability relation with respect to the projection P. Algorithm 1 is used to construct partial automata $\tilde{S}_i = (\tilde{S}_i, \langle \tilde{o}_i, \tilde{t}_i \rangle)$, subautomata of S_i, with $\tilde{t}_i$ denoted by $\rightarrow_{\prime i}$ and the behavior homomorphism $\tilde{l}_i : \tilde{S}_i \rightarrow \mathcal{L}$. According to Theorem 5.7 $\tilde{S}_i$ represent the "local" $\sup \mathrm{N}(K_i, P_i^{-1}(L_i), P)$.

This enables us to consider the behaviors of the corresponding output automata $\tilde{S}$ and $\tilde{S}_i$ of Algorithm 1 with the corresponding parameters detailed above. We show that

$$R = \left\{ \left\langle [\tilde{l}(s_0)]_w, \left[\bigcap_{i=1}^{n} \tilde{l}_i(s_0^i) \right]_w \right\rangle \mid w \in (\tilde{l}(s_0)) \right\}$$

is a bisimulation relation, from which the claim of the theorem follows by coinduction. Take a $w \in (\tilde{l}(s_0))$ arbitrary, but fixed. The items (i)–(iii) below refer to the definition of a bisimulation stated in Section 3.

(i) is trivial: marking is not considered.

(ii) Let $[\tilde{l}(s_0)]_w \xrightarrow{a}$ for $a \in A$, i.e. $(s_0)_w \xrightarrow{a}{}'$. Thus, according to step 2 of Algorithm 1 $(s_0)_w \xrightarrow{a}_1$ and condition (*) of Algorithm 1 is satisfied. Thus, we know that $wa \in L$ and $wa \in (K \cap L)$. It must be shown that $[\cap_{i=1}^n \tilde{l}_i(s_0^i)]_w \xrightarrow{a}$. We need to show that for any $i \in \mathbb{Z}_n$ we have $[\tilde{l}_i(s_0^i)]_w \xrightarrow{a}$, i.e. $(s_0^i)_w \xrightarrow{a}_{\prime i}$. According to Algorithm 1 this amounts to show that for any $i \in \mathbb{Z}_n$ we have $(s_0^i)_w \xrightarrow{a}_{1i}$ and condition (*) of Algorithm 1 holds. Notice that for any $i \in \mathbb{Z}_n$ we have $(s_0^i)_w \xrightarrow{a}_{1i}$, because $wa \in (K \cap L) = \cap_{i=1}^n K_i$. Next we show that condition (*) of Algorithm 1 (with appropriate parameters) is satisfied. If $a \in A_u \cup A_{uo}$ then according to condition (*) of Algorithm 1 it must be checked that $\forall v \in (A_u \cup A_{uo})^*$: $(s_0^i)_{wa} \xrightarrow{v}_i \Rightarrow (s_0^i)_{wa} \xrightarrow{v}_{1i}$. Let $v \in (A_u \cup A_{uo})^*$: $(s_0^i)_{wa} \xrightarrow{v}_i$. This is equivalent to $wav \in P_i^{-1}(L_i)$. We need to show that $(s_0^i)_{wa} \xrightarrow{v}_{1i}$. The structural conditions of modular controllability (MC) and modular normality (MN) will be used. It will be shown by structural induction along the string $v \in (A_u \cup A_{uo})^*$ that $wav \in L$. It is clear that $v = v_1 \dots v_k$ for some $k \in \mathbb{N}$ with $v_m \in A_u \cup A_{uo}$, $m \in \mathbb{Z}_k$. The base step is trivial: $wa\varepsilon \in L$. The induction hypothesis is that $wav_1 \dots v_{m-1} \in L$. We show that $wav_1 \dots v_m \in L$ as well using GMC and GMN. We assume that $wav_1 \dots v_{m-1} \in L$. Thus, if $v_m \in A_u$, then using MC $wav_1 \dots v_{m-1} v_m \in LA_u \cap P_i^{-1}(L_i) \subseteq L$. If $v_m \in A_{uo}$ (i.e. $P(v_m) = \varepsilon$) then MN is applied: $P(wav_1 \dots v_{m-1} v_m) = P(wav_1 \dots v_{m-1})$, implies using MN $wav_1 \dots v_{m-1} v_m \in P^{-1}P(L) \cap P_i^{-1}(L_i) \subseteq L$. By induction we have $wav \in L$. This is equivalent to $(s_0)_{wa} \xrightarrow{v}$. Since $a \in A_u \cup A_{uo}$ and $v \in (A_u \cup A_{uo})^*$, it follows from $(s_0)_w \xrightarrow{a}{}'$ and condition (*) of Algorithm 1 that $(s_0)_{wa} \xrightarrow{v}_1$. This is equivalent to $wav \in (K \cap L)$. Since $K \cap L = \bigcap_{i=1}^n K_i$ we have also $wav \in K_i$. This shows that $(s_0^i)_{wa} \xrightarrow{v}_{1i}$.

If $a \in A_c \cap A_o$ then according to condition (*) of Algorithm 1 it must be checked that $\forall s' \approx_{\mathrm{Aux}(S_i)} (s_0^i)_w : s' \xrightarrow{a}_i \Rightarrow s' \xrightarrow{a}_{1i}$, in which case also $\forall v \in (A_u \cup A_{uo})^*$: $s'_a \xrightarrow{v}_i \Rightarrow s'_a \xrightarrow{v}_{1i}$. Let $s' \approx_{\mathrm{Aux}(S_i)} (s_0^i)_w : s' \xrightarrow{a}_i$. According to the second part of Lemma 5.5 (S_i is a state-partition automaton) there exists $r \in A^*$ such that $P(w) = P(r)$ and $s' = (s_0^i)_r$. Thus, $s' \xrightarrow{a}_i$ is equivalent to $ra \in P_i^{-1} L_i$. It must be shown that $s' \xrightarrow{a}_{1i}$, i.e. $ra \in K_i$. We recall first that $wa \in L$ and $P(wa) = P(ra)$. Hence, $ra \in P^{-1}P(L) \cap P_i^{-1}(L_i) \subseteq L$. Thus, we have $ra \in L$, which is equivalent to $(s_0)_r \xrightarrow{a}$. Since by Lemma 5.5 $(s_0)_r \approx_{\mathrm{Aux}(S)} (s_0)_w$, condition (*) of Algorithm 1 (applied to $(s_0)_r$ playing the role of s') implies that $(s_0)_r \xrightarrow{a}_1$, which is equivalent to $ra \in (K \cap L)$. Since for any $i \in \mathbb{Z}_n$: $(K \cap L) \subseteq K_i$, we have $s' \xrightarrow{a}_{1i}$, which was to be shown. The rest is similar as for $a \in A_u \cup A_{uo}$: the end of the second part of condition (*) of Algorithm 1 is similar to the first part of condition (*) of Algorithm 1. The inductive application of MC and MN shows that $\forall v \in (A_u \cup A_{uo})^*$: $s'_a \xrightarrow{v}_i \Rightarrow s'_a \xrightarrow{v}_{1i}$. We conclude that $(s_0^i)_w \xrightarrow{a}_{\prime i}$ for any $i \in \mathbb{Z}_n$, i.e. $[\cap_{i=1}^n \tilde{l}_i(s_0^i)]_w \xrightarrow{a}$.

(iii) Let $\cap_{i=1}^n \tilde{l}_i(s_0^i)_w \xrightarrow{a}$, i.e. for any $i \in \mathbb{Z}_n$ we have $[\tilde{l}_i(s_0^i)]_w \xrightarrow{a}$, or equivalently, $(s_0^i)_w \xrightarrow{a}_{\prime i}$. We must show that $[\tilde{l}(s_0)]_w \xrightarrow{a}$ for $a \in A$, i.e. $(s_0)_w \xrightarrow{a}{}'$. It must be checked that $(s_0)_w \xrightarrow{a}_1$ and condition (*) of Algorithm 1 applied to automata S and T is satisfied. Firstly, according to step 2 of Algorithm 1 we must show that $(s_0)_w \xrightarrow{a}_1$. But this is straightforward: from $\forall i \in \mathbb{Z}_n$ $(s_0^i)_w \xrightarrow{a}_{\prime i}$ we have in particular $(s_0^i)_w \xrightarrow{a}_i$ and $(s_0^i)_w \xrightarrow{a}_{1i}$. Equivalently, $\forall i \in \mathbb{Z}_n$:

$wa \in P_i^{-1}(L_i)$ and $wa \in K_i$. Thus, we have $wa \in L = \bigcap_{i=1}^n P_i^{-1}(L_i)$ and $wa \in (K \cap L) = \bigcap_{i=1}^n K_i$, which means that $(s_0)_w \xrightarrow{a}$ and $(s_0)_w \xrightarrow{a}_1$. Secondly, we must show that condition (*) of Algorithm 1 holds: if $a \in (A_u \cup A_{uo})$ then $\forall u \in (A_u \cup A_{uo})^*$: $(s_0)_{wa} \xrightarrow{u} \Rightarrow (s_0)_{wa} \xrightarrow{u}_1$; and if $a \in A_o \cap A_c$ then $\forall s' \approx_{\mathrm{Aux}(S)} (s_0)_w : s' \xrightarrow{a} \Rightarrow s' \xrightarrow{a}_1$, in which case also $\forall u \in (A_u \cup A_{uo})^*$: $s'_a \xrightarrow{u} \Rightarrow s'_a \xrightarrow{u}_1$. Let $a \in (A_u \cup A_{uo})$ and $u \in (A_u \cup A_{uo})^*$: $(s_0)_{wa} \xrightarrow{u}$. Hence, we have $wau \in L$, i.e. $\forall i \in \mathbb{Z}_n$: $wau \in P_i^{-1}(L_i)$. The last statement is equivalent to $\forall i \in \mathbb{Z}_n$: $(s_0^i)_{wa} \xrightarrow{u}_i$. Our assumption that $\forall i \in \mathbb{Z}_n$: $(s_0^i)_w \xrightarrow{a}_{'i}$ implies according to condition (*) of Algorithm 1 that $(s_0^i)_{wa} \xrightarrow{u}_i$ implies $(s_0^i)_{wa} \xrightarrow{u}_{1i}$, which is equivalent to $wau \in K_i$. Hence $wau \in (K \cap L) = \bigcap_{i=1}^n K_i$, which is equivalent to $(s_0)_{wa} \xrightarrow{u}_1$.

Now, let $a \in A_o \cap A_c$ and $s' \approx_{\mathrm{Aux}(S)} (s_0)_w : s' \xrightarrow{a}$. According to condition (*) of Algorithm 1 we must show that $s' \xrightarrow{a}_1$ and $\forall u \in (A_u \cup A_{uo})^*$: $s'_a \xrightarrow{u} \Rightarrow s'_a \xrightarrow{u}_1$. It follows from Lemma 5.5 (applied to S) and $s' \approx_{\mathrm{Aux}(S)} (s_0)_w$ that there exists $w' \in A^*$ such that $P(w') = P(w)$ and $s' = (s_0)_{w'} \in S$. Hence, $s'_a \xrightarrow{u}$ is equivalent to $w'a \in L = \bigcap_{i=1}^n P_i^{-1}(L_i)$. An application of Lemma 5.5 to S_i yields also $(s_0^i)_{w'} \approx_{\mathrm{Aux}(S_i)} (s_0^i)_w$ (note that $\mathrm{Aux}(S_i)$ is based on the same projection P as $\mathrm{Aux}(S)$). We notice that $w'a \in L \subseteq P_i^{-1}(L_i)$. Therefore $(s_0^i)_{w'} \xrightarrow{a}$, i.e. according to condition (*) of Algorithm 1 (with S_i playing the role of S and T_i playing the role of T) $(s_0^i)_{w'} \xrightarrow{a}_{1i}$. The last statement is equivalent to $w'a \in K_i$. Therefore we have also $w'a \in \cap_{i=1}^n K_i = (K \cap L)$. But this is equivalent to $s' \xrightarrow{a}_1$, which was to be shown. The rest is similar to the case $a \in (A_u \cup A_{uo})$: if $u \in (A_u \cup A_{uo})^*$: $s'_a \xrightarrow{u}$, then $w'au \in L$ with the same w' as above. Thus, $\forall i \in \mathbb{Z}_n$: $w'au \in P_i^{-1}(L_i)$. The last statement is equivalent to $\forall i \in \mathbb{Z}_n$: $(s_0^i)_{w'a} \xrightarrow{u}_i$. Our assumption that $\forall i \in \mathbb{Z}_n$: $(s_0^i)_w \xrightarrow{a}_{'i}$ implies according to the second part of condition (*) of Algorithm 1 (with $(s_0^i)_{w'}$ playing the role of s') that $(s_0^i)_{w'a} \xrightarrow{u}_i$ (meaning in Algorithm 1 $s'_a \xrightarrow{u}_i$) implies $(s_0^i)_{w'a} \xrightarrow{u}_{1i}$, which is equivalent to $w'au \in K_i$. Hence $w'au \in (K \cap L)$, which is equivalent to $s'_a \xrightarrow{u}_1$. The conclusion is $(s_0)_w \xrightarrow{a}_{'}$, which is equivalent to $[\tilde{l}(s_0)]_w \xrightarrow{a}$.

Now we will prove the necessity part. The necessity of modular controllability and modular normality is easy to show along the same lines as in the complete observation case. Let L_i, $i \in \mathbb{Z}_n$ be given and let Eq. (34) hold for any $K \subseteq A^*$. Then in particular (the more difficult) inclusion

$$\sup \mathrm{CN}(K \cap L, L, P, A_u) \subseteq \bigcap_{i=1}^n \sup \mathrm{CN}(K_i, P_i^{-1}(L_i), P, A_u)$$

holds for any $K \subseteq A^*$. This is equivalent to

$$\sup \mathrm{CN}(K \cap L, L, P, A_u) \subseteq \sup \mathrm{CN}(K_i, P_i^{-1}(L_i), P, A_u),\ K \subseteq A^* \text{ and } i \in \mathbb{Z}_n.$$

The inclusion holds in particular for $K = L$, which entails $K_i = L$ for any $i \in \mathbb{Z}_n$. Hence, $L = \sup \mathrm{CN}(K \cap L, L, P, A_u) \subseteq \sup \mathrm{CN}(L, P_i^{-1}(L_i), P, A_u)$. Since $\sup \mathrm{CN}(L, P_i^{-1}(L_i), P, A_u) \subseteq L$ by definition of the sup C operator, we obtain $L = \sup \mathrm{CN}(L, P_i^{-1}(L_i), P, A_u)$. But this means that L is for any $i \in Z_n$ both controllable with respect to $P_i^{-1}(L_i)$ and A_u and $(P_i^{-1}(L_i), P)$-normal, which are the modular controllability and the modular normality conditions. □

Note that global mutual controllability together with global mutual normality imply modular controllability and modular normality, and the former notions are easier to verify than the latter ones, because they do not include the global plant. In fact, although modular controllability and modular normality can be checked in polynomial time (but in size of global DES!), their verification requires the construction of the global system, which contradicts the modular approach, because the size of global system grows exponentially with the number of components. On the other hand verification of global mutual controllability and global mutual normality is polynomial in the size of the local components. A polynomial algorithm for checking observability (can be adapted for normality) has been presented in [24]. See Section 8.4 for comments on the computational time complexity of both global and modular control synthesis in this case.

7.2. *Case of supremal normal sublanguage*

As a special case of Theorem 7.4 we have.

Corollary 7.5. *Modular control synthesis equals global control synthesis for the supremal normal sublanguage in the case of a distributed DES and of an indecomposable specification. Consider a distributed DES. Assume that the local plants agree on the observability of their common events.*

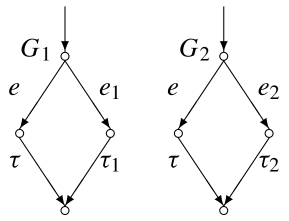
If the local plant languages $\{L_i,\ i \in Z_n\}$ are modularly normal then for any indecomposable specification $K \subseteq A^*$

$$\sup \mathrm{N}(K \cap L, L, P) = \bigcap_{i=1}^{n} \sup \mathrm{N}(K_i, P_i^{-1}(L_i), P). \tag{35}$$

Conversely, if for a given modular plant equality (35) holds for any global specification $K \subseteq A^*$ then the local plant (partial) languages $\{L_i,\ i \in Z_n\}$ are modularly normal.

Now we present an example, where it is shown that global mutual normality (GMN) is not a necessary condition.

Example 7.6. Let $A = \{e, e_1, a_2, \tau, \tau_1, \tau_2\}$, $A_1 = \{e_1, \tau_1, e, \tau\}$, $A_2 = \{e_2, \tau_2, e, \tau\}$, $A_o = \{e_1, e_2, e\}$, $A_{o,1} = \{e_1, e\}$, and $A_{o,2} = \{e_2, e\}$. Consider the local plant languages (the marked languages are not considered) displayed in the figure below.



Let prefix$(K) = \{\varepsilon, e, e_1, e_1e_2\}$. One can easily verify that K is not decomposable. Indeed, the inclusion prefix$(K) \subset P_1^{-1}P_1(\text{prefix}(K)) \cap P_2^{-1}P_2(\text{prefix}(K))$ is strict, i.e. prefix$(K) \neq P_1^{-1}P_1(\text{prefix}(K)) \cap P_2^{-1}P_2(\text{prefix}(K))$. Computing further the parallel product $L = L_1 \parallel L_2$ yields:

$$\sup \mathrm{N}(K \cap L, L, P) = \{\varepsilon\}.$$

Note that in this example $K_i = K \cap P_i^{-1}L_i = K$ for $i = 1, 2$. It is also easy to see that $\sup \mathrm{N}(K_i, P_i^{-1}(L_i), P) = \{\varepsilon\}$ as well for $i = 1, 2$. i.e. the commutativity holds trivially true. On the other hand, global mutual normality does not hold. We have e.g.

$$\tau_1 \in (P^{-1}PP_1^{-1})(L_1) \cap (P_2)^{-1}(L_2) \setminus P_1^{-1}(L_1).$$

This only shows that global mutual normality is not a necessary condition (and not necessary structural condition), because for another specification $K' = \{\varepsilon, e, e\tau, e_1, e_1e_2\}$ notice that $e\tau \in K' \setminus K$ we have also $K_i' = K' \cap P_i^{-1}(L_i) = K'$, $i = 1, 2$, thus $\sup \mathrm{N}(K' \cap L, L, P) = \{\varepsilon, a, a\tau\}$, while still $\sup \mathrm{N}(K_i', P_i^{-1}(L_i), P) = \{\varepsilon\}$. Therefore, for K', modular and global computation of the (global) supremal normal sublanguage differ, which is not in contradiction with Theorem 7.5, because we know that global mutual normality does not hold for this example. Note that an example, showing that GMN is not necessary among structural conditions, requires that $n \geq 3$, similarly as for GMC.

7.3. Antipermissive control policy

In this subsection our intention is to generalize the results concerning modular computation of supremal normal sublanguages of global specifications to modular computation of closed-loop languages using the antipermissive control policy. This will be done in the very same way as was done in Section 4 for local specification. Unlike the setting of Section 4 we will work with $P_i^{-1}(L_i)$ instead of L_i. Since we work with the global alphabet, synchronous composition coincides with intersection. The concept of modular observability using Definition 2.5 is needed:

Definition 7.7. Consider a distributed DES. The modular plant languages $\{L_i, i \in Z_n\}$ are called *modularly observable* if for any $i \in Z_n$ we have that L is observable with respect to $P_i^{-1}(L_i)$ and A_o.

Theorem 7.8. *Modular control synthesis equals global control synthesis for the closed-loop language in the case of a distributed DES, of an indecomposable specification, and of an antipermissive control policy. Consider a distributed discrete-event system. Assume that the local plants* (1) *agree on the controllability of their common events and* (2) *agree on the observability of their common events.*

If $\{L_i,\ i \in Z_n\}$ are modularly observable then for any indecomposable specification $K \subseteq A^*$

$$\mathrm{AP}(K \cap L, L, P) = \bigcap_{i=1}^{n} \mathrm{AP}(K_i, P_i^{-1}(L_i), P). \tag{36}$$

Conversely, if for a given modular plant equality (36) *holds for any global specification K then the local plant (partial) languages $\{L_i,\ i \in Z_n\}$ are modularly observable.*

Proof. The sufficiency part can be shown by coinduction using the same algorithm for closed-loop language under antipermissive control policy as was done in [6] in the case of local specification languages. The proof is very similar to the proof of Theorem 7.4. In view of our previous results it is straightforward to show that a necessary structural condition for local antipermissive control policy to equal global antipermissive control policy is that $\mathrm{AP}(L, P_i^{-1}(L_i), P) = L$, which means that modular observability holds. □

A sufficient condition can be formulated in terms of global mutual observability, which is similar to strong global mutual normality, except that observability plays the role of normality. Similarly as for normality, global mutual observability is then computationally much easier to verify than modular observability.

8. Computational complexity of modular control synthesis

The aim of this section is to illustrate the interest of modular control synthesis by comparing the computational complexity of modular control synthesis with the global control synthesis. This will be done for several control problems studied in the previous sections.

8.1. Computational complexity in the case of a decomposable specification

We start with Section 3 and the synthesis of modular supervisors with full observations, i.e. supremal controllable sublanguages that are used. The following symbols will be used.

$n_m \in \mathbb{N}$	number of modules,
n_i	size of the minimal state set of a recognizer of module $i \in \mathbb{Z}_{n_m}$,
$n^* =$	$\max_{i\in\mathbb{Z}_{n_m}} n_i \in \mathbb{N}$,
m_i	size of the event set of module $i \in \mathbb{Z}_{n_m}$,
$m^* =$	$\max_{i\in\mathbb{Z}_{n_m}} m_i, \quad m_* = \min_{i\in\mathbb{Z}_{n_m}} m_i$,
$r^* =$	$\max_{i,j\in\mathbb{Z}_{n_m}} \lVert A_i \cap A_j \rVert \in \mathbb{N}$,
$n_L =$	size of the minimal state set of the recognizer of the global plant,
$m =$	size of the event set of the global system,
k_i	size of the minimal state set of a recognizer $i \in \mathbb{Z}_{n_m}$, of the local specification $K_i \subseteq A_i^*$,
$k^* =$	$\max_{i\in\mathbb{Z}_{n_m}} k_i \in \mathbb{N}$,
m_K	size of the event set of the minimal recognizer of the specification,
n_K	size of the minimal state set of the recognizer of the specification.

We have the following simple inequalities and bounds:

$$n_L \le \prod_{i=1}^{n_m} n_i \le (n^*)^{n_m}, \tag{37}$$

$$n_K \leq \prod_{i=1}^{n_m} k_i \leq (k^*)^{n_m} \tag{38}$$

$$m \leq \sum_{i=1}^{n_m} m_i \leq (m^*)^{n_m}. \tag{39}$$

It was shown in [1, p. 115] that the time complexity of the computation of the supremal controllable sublanguage of the global plant in terms of the minimal size of the state set n and the minimal size of the specification recognizer n_K is

$$O(n_L n_K^2). \tag{40}$$

We also recall that it has been shown in [13] that for prefix-closed specification the time complexity of the computation of the supremal controllable sublanguage of the global plant is of the order $O(n_L n_K)$, which is the basis for our estimates. This is also the complexity for checking the controllability of the specification. Hence,we have

Proposition 8.1. *The computational time complexity of the global computation of the optimal supervisor (represented by the supremal controllable sublanguage) is given by*

$$O(n_L n_K) \leq O((n^*)^{n_m}(k^*)^{n_m}). \tag{41}$$

The computation of modular control synthesis proceeds by the followings steps:

Algorithm 2. (1) Check whether the modules agree on the controllability of their common events:

$$A_{iu} \cap A_j = A_i \cap A_{ju}, \forall i, j \in \mathbb{Z}_{n_m}. \tag{42}$$

(2) Test whether mutual controllability holds: i.e.

$$\text{prefix}(L_j)(A_{ju} \cap A_i) \cap P_j(P_i)^{-1}\text{prefix}(L_i) \subseteq \text{prefix}(L_j), \forall i, j \in \mathbb{Z}_{\text{n}}, i \neq j. \tag{43}$$

(3) For each module compute the local supremal controllable sublanguage:

$$\sup \mathrm{C}(K_i, L_i, A_{iu}),\ \forall i \in \mathbb{Z}_{n_m}. \tag{44}$$

Proposition 8.2. *Consider the discrete-event system defined above.*

(a) *The time complexity of checking the agreement on the controllability of their common events is*

$$O(n_m(n_m - 1)m^*) = O(n_m^2 m^*). \tag{45}$$

(b) *The time complexity of the check of mutual controllability is*

$$O\left(\sum_{i=1}^{n_m}\sum_{j=1}^{n_m}(2^{n_i})n_j\right) = O(n_m^2 2^{n^*} n^*). \tag{46}$$

Note that the term 2^{n_i} appears, because $P_i P_j^{-1}(L_j)$ must be computed and the natural projections are computed with exponential worst case complexity, although in most cases it can be calculated much faster (cf. [31]).

(c) *The time complexity of the computation of all local supremal controllable sublanguages is*

$$\sup \mathrm{C}(K_i, L_i, A_{iu}) \quad O(n_i k_i), \tag{47}$$

$$\{\sup \mathrm{C}(K_i, L_i, A_{iu}), i \in \mathbb{Z}_{n_m}\} \quad O\left(\sum_{i=1}^{n_m} n_i k_i\right) = O(n_m n^* k^*). \tag{48}$$

(d) *The time complexity of the computation of the supremal controllable sublanguage by modular control synthesis as described in Section 4 is*

$$O(n_m^2 2^{n^*} k^* m^*). \tag{49}$$

Proof (d).

$$O(n_m^2 m^*) + O(n_m^2 2^{n^*} n^*) + O(n_m n^* k^*) \leq O(n_m^2 2^{n^*} k^* m^*). \quad \square$$

8.2. Time complexity of the computation of the supremal controllable language in the case of a global specification

Algorithm 3. Computation of the global specification. Let us denote by n_K the size of the minimal state set of the recognizer of the global specification. The remaining notation is as introduced before.

(1) Compute the intersection of the specification and the plant (denoted resp. G_K and G_L),

$$K \cap L = L(G_K) \cap L(G_L) = L(G_{K \cap L}) = L(G_k \times G_L). \tag{50}$$

(2) Compute the supremal controllable sublanguage according to the algorithm described in [1].

$$\sup \mathrm{C}(K \cap L, L, A_u). \tag{51}$$

Proposition 8.3. *The computational time complexity of the supremal controllable sublanguage of the global plant in the case of an indecomposable global specification.*

(a) *The computational time complexity of the computation of the intersection of the specification and the plant.*

$$O(n_L n_K). \tag{52}$$

(b) *The computational time complexity of the computation of the supremal controllable sublanguage of the global system.*

$$O(n_L.n_L.n_K) = O(n_L^2 n_K) \le O((n^*)^{2n_m} n_K). \tag{53}$$

8.3. The time complexity of the modular computation of the supremal controllable sublanguage in the case of an indecomposable global specification

Algorithm 4. The computation of the supremal controllable sublanguage according to modular control synthesis proceeds by the followings steps:

(1) Check whether the modules agree on the controllability of their common events:

$$A_{iu} \cap A_j = A_i \cap A_{ju}, \forall i, j \in \mathbb{Z}_{n_m}. \tag{54}$$

(2) Compute the global inverse images of the modules:

$$P_i^{-1}(L_i), \forall i \in \mathbb{Z}_{n_m}. \tag{55}$$

This is done by adding at every state of the recognizer of L_i selfloops for the events in $A \backslash A_i$.

(3) Test whether global mutual controllability holds

$$P_j^{-1}(L_j)A_{ju} \cap P_i^{-1}(L_i) \subseteq P_j^{-1}(L_j), \forall i, j \in \mathbb{Z}_{n_m},\ i \neq j. \tag{56}$$

(4) Computation of the over approximation of the local specification,

$$K_i = K \cap P_i^{-1}(L_i), \forall i \in \mathbb{Z}_{n_m}. \tag{57}$$

This is done by computing the product automaton

$$L(G_{K_i}) = L(G_K \cap P_i^{-1}(L_i)) = L(G_K \times G_{P_i^{-1}(L_i)}). \tag{58}$$

(5) For each module compute the local supremal controllable sublanguage:

$$\sup \mathrm{C}(K_i, P_i^{-1}(L_i), A_u), \forall i \in \mathbb{Z}_{n_m}. \tag{59}$$

(6) Computation of the intersection

$$\bigcap_{i=1}^{n} \sup \mathrm{C}(K_i, P_i^{-1}(L_i), A_u). \tag{60}$$

Proposition 8.4. *Consider the discrete-event system defined above.*

(a) *The time complexity of checking the agreement on the controllability of their common events is*

$$O(n_m(n_m-1)m^*) = O(n_m^2 m^*). \tag{61}$$

(b) *The time complexity of computing the global inverse images of the modules is*

$$P_i^{-1}(L_i) \quad O(n_i(m-m_i)) \le O(n_i m); \tag{62}$$

$$\{P_i^{-1}(L_i), i \in \mathbb{Z}_{n_m}\} \quad O\left(\sum_{i=1}^{n_m} n_i(m-m_i)\right) \le O(n_m n^* m). \tag{63}$$

(c) *The time complexity of the over approximation of the local specification,*

$$K_i = K \cap P_i^{-1}(L_i) \quad O(n_i n_K m); \tag{64}$$

$$\{K_i, i \in \mathbb{Z}_{n_m}\} \quad O\left(\sum_{i=1}^{n_m} n_i n_K m\right) = O(n_m n^* n_K m). \tag{65}$$

(d) *The time complexity of the check of global mutual controllability is*

$$O\left(\sum_{i=1}^{n_m}\sum_{j=1}^{n_m} n_i m n_j m\right) = O(n_m^2 (n^*)^2 m^2). \tag{66}$$

(e) *The time complexity of the computation of all local supremal controllable sublanguages is*

$$\sup \mathrm{C}(K_i, L_i, E_u) \quad O(n_i n_i n_K m) = O(n_i^2 n_K m), \tag{67}$$

$$\{\sup \mathrm{C}(K_i, L_i, E_u), i \in \mathbb{Z}_{n_m}\} \quad O\left(\sum_{i=1}^{n_m} n_i^2 n_K m\right) = O(n_m (n^*)^2 n_K m). \tag{68}$$

(f) *The time complexity of the computation of the supremal controllable sublanguage by modular control synthesis as described in this paper is*

$$O(n_m^2 (n^*)^2 n_K^2 m^2). \tag{69}$$

Proof (f).

$$O(n_m^2 m^*) + O(n_m n^* m) + O(n_m n^* n_K m) + O(n_m^2 (n^*)^2 m^2) + O(n_m (n^*)^2 n_K m) \le O(n_m^2 (n^*)^2 n_K m^2). \quad \square$$

8.4. The time complexity for the computation of the supremal controllable and normal sublanguage

The time complexity of the computation of the supremal controllable and normal sublanguage is stated in [1] as being exponential in the size of a minimal recognizer and of the size of the minimal recognizer of the specification language. The same expression is used in the paper [13]. The formulas need to be used to derive an explicit expression for the time complexity. Below is used the following formula

$$O(2^{n_L \cdot n_K}). \tag{70}$$

It follows that the complexity of the global computation is double exponential: $O(2^{(n^*)^{2n_m} \cdot n_K})$ in the case of a global specification and $O(2^{(n^*)^{n_m} \cdot (k^*)^{n_m}})$ in the case of local specifications.

On the other hand, modular computation is only single exponential in the size of local plants and specifications. The main step: computation of local supremal controllable and normal sublanguage takes $O(2^{(n^*)^2 \cdot n_K})$ time for global specification and $O(2^{n^* \cdot k^*})$ for local specifications. One can verify that together with checking all sufficient conditions, for a large number of modules it is much more advantageous to apply modular control synthesis method over global synthesis method.

9. Conclusion

An overview of the main concepts and results have been presented for modular control synthesis to equal global control synthesis for control of discrete-event systems together with several new results. Most of the possible special cases of modular and distributed control have been covered (both fully and partially observed systems, both local and global specifications). Among the new results, we have provided for all cases necessary conditions for modular control synthesis to equal global control synthesis. Antipermissive control policy was proposed for the global specification and necessary and sufficient conditions have been presented for global antipermissive control policy to yield the same result as computationally much more efficient local antipermissive control policy. It was shown that in the shuffle case of a distributed DES with a decomposable specification, where mutual normality fails, this condition is actually not needed for modular control synthesis to equal global control synthesis.

These results are important for the optimal supervisory control of large distributed plants, because with respect to the derived conditions control synthesis can be implemented modularly. All the sufficient conditions we have presented have a lower computational time complexity than their counterparts for global systems. The structural conditions do not depend on the specification, which is very important for global (indecomposable) specifications.

Acknowledgements

The work of Jan Komenda was partially supported by the Grant GA AV No. B100190609 and of the Academy of Sciences of the Czech Republic, Institutional Research Plan No. AV0Z10190503.

References

[1] R.D. Brandt, V. Garg, R. Kumar, F. Lin, S.I. Marcus, W.M. Wonham, Formulas for calculating supremal controllable and normal sublanguages, Systems & Control Letters 15 (1990) 111–117.

[2] S.G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Kluwer Academic Publishers, Dordrecht, 1999.

[3] B. Gaudin, H. Marchand, Modular supervisory control of a class of concurrent discrete event systems, in: Proceedings WODES'04, Workshop on Discrete-Event Systems, Reims, 22–24 September, 2004, pp. 181–186.

[4] J. Komenda, J.H. van Schuppen, Supremal normal sublanguages of large distributed discrete-event systems, in: Proceedings WODES'04, Workshop on Discrete-Event Systems, Reims, 22–24 September, 2004.

[5] J. Komenda, Modular control of large distributed discrete-event systems with partial observations, in: Proceedings of the 15th International Conference on Systems Science, vol. II, Wroclaw, Poland, September 2004, pp. 175–184.

[6] J. Komenda, J.H. van Schuppen, Modular antipermissive control of discrete-event systems, in: Proceedings of IFAC World Congress 2005, Prague, July 2005.

[7] J. Komenda, J.H. van Schuppen, Supremal sublanguages of general specification languages arising in modular control of discrete-event systems, in: Proceedings of Joint 44th IEEE Conference on Decision and Control and European Control Conference, Sevilla, December 2005, pp. 2775–2780.

[8] J. Komenda, J.H. van Schuppen, B. Gaudin, H. Marchand, Modular supervisory control with general indecomposable specification languages, in: Proceedings of Joint 44th IEEE Conference on Decision and Control and European Control Conference, Sevilla, 12-15.12, 2005.

[9] J. Komenda, J.H. van Schuppen, Modular control of discrete-event systems with coalgebra, Accepted for publication in IEEE Transactions on Automatic Control, to appear in April 2008 issue.

[10] J. Komenda, J.H. van Schuppen, Control of discrete-event systems with partial observations using coalgebra and coinduction, Discrete Event Dynamical Systems: Theory and Applications 15 (3) (2005) 257–315.

[11] J. Komenda, J.H. van Schuppen, Control of modular and distributed discrete-event systems, in: Proceedings of 4th Symposium on Formal Methods in Components and Objects, FMCO, in: LNCS, vol. 4111, Springer, 2006.

[12] J. Komenda, J.H. van Schuppen, Optimal solutions of modular supervisory control problems with indecomposable specification languages, in: Proceedings of the 8th International Workshop on Discrete Event Systems, WODES, IEEE Press, New York, 2006, pp. 143–148.

[13] R. Kumar, V.K. Garg, S.I. Marcus, On controllability and normality of discrete event dynamical systems, Systems & Control Letters 17 (1991) 157–168.

[14] P. Madhusudan, P.S. Thiagarajan, Branching time controllers for discrete event systems, Theoretical Computer Science 274 (2002) 117–149.

[15] M. Lamouchi, J.G. Thistle, Effective control synthesis for discrete event systems under partial observations, in: Proceedings of the IEEE Conference on Decision and Control, 2000.

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

[17] A. Pnueli, R. Rosner, Distributed reactive systems are hard to synthesize, in: Proceedings of the 1990 IEEE Symposium on the Foundations of Computer Science, IEEE, New York, 1990, pp. 746–757.

[18] K. Rohloff, S. Lafortune, The control and verification of similar agents operating in a broadcast network environment, in: Proceedings CDC 2003, Hawaii, USA.

[19] K. Rohloff, S. Lafortune, Recent results on computational issues in supervisory control, in: Proceedings of the ATPN Workshop on Discrete Event Systems Control, Eindhoven, The Netherlands, June 2003, pp. 15–35.
[20] P.J. Ramadge, W.M. Wonham, The control of discrete-event systems, Proceedings of the IEEE 77 (1989) 81–98.
[21] J.J.M.M. Rutten, Coalgebra, concurrency, and control, Research Report CWI, SEN-R9921, Amsterdam, November 1999. Available also at: http://www.cwi.nl/˜janr.
[22] J.J.M.M. Rutten, Universal coalgebra: A theory of systems, Theoretical Computer Science 249 (1) (2000) 3–80.
[23] S. Tripakis, Undecidable problems of decentralized observation and control, in: Proceedings of the IEEE Conference on Decision and Control, 2001.
[24] J.N. Tsitsiklis, On the control of discrete-event dynamical systems, Mathematics of Control, Signal, and Systems (1989) 95–107.
[25] K.C. Wong, S. Lee, Structural decentralized control of concurrent discrete-event systems, European Journal of Control 8 (2002) 477–491.
[26] I. Walukiewicz, A. Arnold, A. Vincent, Games for synthesis of controlers with partial observation, Theoretical Computer Science 303 (1) (2003) 7–34.
[27] W.M. Wonham, Lecture notes on control of discrete-event systems, University of Toronto, Department ECE, Toronto, 2005. http://www.control.toronto.edu/people/profs/wonham/wonham.html.
[28] Y. Willner, M. Heymann, Supervisory control of concurrent discrete-event systems, International Journal of Control 54 (1991) 1143–1166.
[29] W.M. Wonham, P.J. Ramadge, On the supremal controllable sublanguage of a given language, SIAM Journal on Control and Optimization 25 (1987) 637–659.
[30] W.M. Wonham, P.J. Ramadge, Modular supervisory control of discrete-event processes, Mathematics of Control, Signal and Systems 1 (1988) 13–30.
[31] K.C. Wong, On the complexity of projections of discrete-event systems, in: Proc. 4th Int. Workshop Discrete Event Syst, WODES, 1998, pp. 201–206.