Modular Control of Discrete-Event Systems with Coalgebra

Jan Komenda
Institute of Mathematics
Czech Academy of Sciences, Brno Branch,
Zizkova 22, 616 62 Brno, Czech Republic
E-mail: komenda@ipm.cz
and
Jan H. van Schuppen
Centrum voor Wiskunde en Informatica (CWI)
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
E-mail: J.H.van.Schuppen@cwi.nl

Abstract

Modular supervisory control of discrete-event systems (DES), where the overall system is composed of subsystems that are combined in synchronous (parallel) product, is considered. The main results of this paper are formulations of sufficient conditions for the compatibility between the synchronous product and various operations stemming from supervisory control as supervised product and supremal controllable sublanguages. These results are generalized to the case of modules with partial observations: e.g. modular computation of supremal normal sublanguages is studied. Coalgebraic techniques: e.g. coinduction proof principle are used in our main results. It is guaranteed that under the conditions derived in the paper control synthesis can be done locally without affecting safety or optimality of the solution. An algorithmic procedure for checking the new conditions is proposed and the computational benefit of the modular approach is discussed and illustrated by comparing the time complexity of modular and the monolithic computation.

1 Introduction

The purpose of this paper is to develop modular synthesis of discrete-event systems (DES) and to show how coalgebra can be effectively used for its solution.

A short historical overview of modular supervisory control of DES follows. Modular approach to the supervisory control of DES has been introduced by P.J. Ramadge and W.M. Wonham in [34]. The system is composed of local components (subsystems) that run concurrently (in parallel), i.e. the global system is the synchronous product of the local components. In the first papers on the topic, the input alphabets of the local components were identical ([46], [24]). The general case of different local input alphabets has been studied in [43], where a very restrictive condition is imposed on events shared by several local alphabets: they must be controllable for all subsystems. This assumption has been generalized recently in [44] to the condition that the shared events must have the same control status for all subsystems that share a particular event. All the above mentioned references concern only modular control with full observations. Very little attention has been payed so far to the modular control with partial observations. A special case of modular supervisory control with partial observations is studied in [28]. Computational aspects of modular control have been recently studied in [29].

The main problems of modular supervisory control are: Can the supervisor be synthesized at the local level and then be combined to a global supervisor without affecting the optimality of the solution? If the answer to this question is positive, then there is an exponential saving on the computational complexity. In our coalgebraic framework this problem can be paraphrased as follows: when does the supervised product commute with the synchronous product and when does the supremal normal and/or controllable sublanguage commute with the synchronous product? (recall that the synchronous product of partial languages has been defined by coinduction in [31], see also Section 3 and Appendix A).

The modular control problem is formulated in a coalgebraic framework. This allows the use of concepts and theorems of coalgebra thus simplifying proofs and leading to new algorithms. Attention is restricted to modular control synthesis without blocking as the blocking issue requires different concepts and methods. Blocking is regarded as important by the authors and it will receive attention in a possible future publication. For monolithic DES with partial observations (monolithic DES refers to DES without the modular structure to distinguish them from modular DES) the conditions for the existence of nonblocking solutions are the same as in the case of full observations. It is to be expected that for modular DES with partial observations the conditions for nonblocking are also the same as for modular DES with full observations. Section 3 contains preliminaries on coalgebra and coinduction. The reader interested in more details about these concepts may read [31] and [32].

This paper is an extended version of [19], where our results are stated without proofs. The first set of results includes Theorem 4.2 which states a sufficient condition for the property that in modular control with complete observations, the supervised product commutes with the parallel composition operation. Further, Theorem 4.5 states that mutual controllability and a second condition imply the commutativity of the supremal controllable sublanguage with the parallel composition operation. This second result is already known [44], but the proofs we propose rely on the uniform framework of universal coalgebra and in our opinion simplify those of Lee and Wong [44]. Nevertheless, in contrast to this reference our paper does not address the blocking issue, which requires an additional condition, but also different concepts and methods. Thus nonblocking modular control synthesis with complete observations is possible without loss of global optimality if the condition of mutual controllability and two additional conditions hold. Nonblocking modular control is also studied in [12] and in [38], where the hierarchical approach is used.

The second set of results concern modular synthesis if at every local module only partial observations of the local events are available. For this the concept of mutual normality is formulated. Theorem 5.13 establishes a sufficient condition for the commutativity of the supervised product and the parallel composition operation. Theorem 5.21 states that an auxiliary algorithm for the computation of the supremal normal sublanguage of a considered language is correct. Theorem 5.26 then states that if mutual normality and a second condition both hold then there is commutation of the supremal normal sublanguage with the parallel composition.

Section 5 presents two academic examples and an algorithm for checking mutual normality.

2 Problem formulation and approach

The presentation in this section is exclusively verbal, a mathematical framework is developed from Section 3 onwards and the technical details are in the appendix. Readers not familiar with the coalgebraic approach are invited to read section 3 and appendix slowly.

2.1 Modular discrete-event systems and supervisory control

Discrete-event systems (DES) are new types of dynamical systems whose evolution is event triggered as opposed to timed triggered evolution of classical continuous and discrete time systems. Supervisory control developed by P. Ramadge and W.M. Wonham and coworkers is now a well established theory for control of discrete-event systems modelled by automata or Petri Nets. Control problems address the control objectives of safety and liveness [7]. A supervisor restricts the behavior of a discrete-event system such that the control objectives (safety, liveness, ...) are met.

The supervisor affects the plant by disabling a subset of controllable events. The interconnection of the system and the supervisory is called the closed-loop system. The supervisory control problem is then to synthesize a supervisor such that the closed-loop system meets the prespecified control objectives.

In this paper we are interested in modular DES (also called concurrent DES), where the (global) system is composed of local components (modules) that interact with each other. The global system is then the parallel composition (i.e. synchronous product) of local components. The methods used for monolithic DES have a very high complexity when applied to modular DES due to the combinatorial state explosition. A typical modular DES is composed of a very large number of relatively small components, small in size. The main goal is then to propose supervisory control methods that avoid building (and manipulating with) the global system.

2.2 Supervisory control with partial observations

A DES with partial observations is a DES, where not all events are observed and hence not all events are available to the controller (superviser). The events which are observed are called the observable events. The events that are not observed are called unobservable events. Examples of such events are failures of a machine or operations in a communication network where the local events are not communicated to a distant observer station. Supervisory control with partial observations is then to synthesize a supervisor based on partial observations only such that the closed-loop system meets the prespecified control objectives. Control with partial observations is highly relevant to engineering because not all events are observed. In this paper we are interested in the DES with partial observations and modular structure, i.e. modular DES with partial observations, where local modules are themselves partially observed DES.

2.3 Coalgebra

Coalgebra was introduced by S. Eilenberg in a 1965 paper [11]. Algebraists did not consider the concept useful until the appearance of a proof on the existence of a final coalgebra, see [1]. The computer scientist R. Milner has used bisimulation for labelled transition systems since 1980 and this is a special case of coalgebra. Since then bisimulation and coalgebra are extensively used in computer science. Coalgebra has been used on other parts of control and system theory since about 1990, see the papers by R. Grossman, [13]. It is used implicitly in the thesis of E.D. Sontag [39].

Briefly, an algebra can be considered, in terms of category theory, as a map from a functor of a set to the corresponding set. A coalgebra is then defined as a map from a set to a functor of the set. A coalgebra is called *final* if there exists a unique structure preserving map (homomorhism) from every coalgebra to the final coalgebra.

A theorem of coalgebra is that if the coalgebra is final then any bisimulation on the product of two sets implies equality of the sets. Another useful theorem is that one can prove existence of an object via coinduction on final coalgebras. Coinduction corresponds to induction as coalgebra corresponds to algebra.

In this paper coalgebra is used in coinductive definitions and proofs. To keep the paper elementary, no category theory is used at all. The reader need not have a background in coalgebra to read the paper. The only results used are that the existence of a bisimulation on language subsets implies equality of the subsets and that new objects (operations) on languages can be constructed by coinduction, which is described in the appendix.

3 Automata, algebra, and coalgebra

In this section automata are first defined as is done classically. Then coalgebra is formally defined and it is shown how automata can be formulated in coalgebraic terms. With every automaton can be associated the partial language which it generates. The subset of partial languages is then given the structure of an automaton. Finally the concept of coinduction is introduced.

3.1 Automata

Automata were used as models of computation from about the 1950's on. Textbooks on automata theory include [14, 10]. System theory, the basis of control theory, has been inspired by automata theory. Therefore control theory and automata theory have the same basis.

An automaton is a collection of sets and functions,

$$(Q, E, f, q_0, Q_m),$$

where Q is a finite set called the *state set*, E is a finite set called the *event set*, $f : Q \times E \to Q$ is a function called the *transition function*, $q_0 \in Q$ is the *initial state*, and $Q_m \subseteq Q$ is the subset of *marked* states. An automaton operates on a string of events and produces a sequence of states, also called a state trajectory:

$$(q_0, q_1, q_2, \ldots), \;\; q_{i+1} = f(q_i, e_i), \; \forall i \in \mathbb{N}_+ = \{0, 1, 2, \ldots\}.$$

Instead of an automaton one also defines a generator. Recall that the transition function of an automaton, f, is defined for all its arguments. It is therefore also called a *total function*. A *generator* is a collection as an automaton above but the transition function is a partial function, for every $q \in Q$ there exists a subset $E(q) \subseteq E$, in general not equal to E, such that $f(q, e)$ is defined for all $q \in Q$ and $e \in E(q)$. Most examples of engineering systems are actually generators rather than automata. In this paper a generator is also called a partial automaton in line with the terminology used by J.J.M.M. Rutten in [31].

3.2 Algebra and coalgebra

An F-*algebra* is a tuple (U, c) consisting of,

$$\begin{array}{rl} U & \text{a set, called the carrier set,} \\ c : F(U) \to U & \text{the operation of the algebra.} \end{array}$$

A F-*coalgebra* is a tuple (U, c) consisting of,

$$\begin{array}{rl} U & \text{a set, called the carrier set,} \\ c : U \to F(U) & \text{the operation of the coalgebra.} \end{array}$$

Let us denote $1 = \{\emptyset\}$ and $2 = \{0, 1\}$ the set of Booleans. As an example consider the functor $F = (f, h)$

$$Q \mapsto F(Q) = (Q + 1)^E \times 2.$$

Then a DES generator (Q, E, f, q_0, Q_m) can be viewed as $(Q, (f, h))$, i.e. as an F-coalgebra. Note that $h : Q \to 2$ can be identified with Q_m and $f : Q \to (Q + 1)^E$ is an equivalent formulation of the (deterministic) partial transition function.

Consider functor F. (a) A *homomorphism of* F-*coalgebras* from a F-coalgebra (X_1, c_1) to a F-coalgebra (X_2, c_2) is a function $f : X_1 \to X_2$ such that $c_2 \circ f = F(f) \circ c_1$,
or commutativity holds in the diagram,

$$\begin{tikzcd} X_1 \arrow[r, "f"] \arrow[d, "c_1"'] & X_2 \arrow[d, "c_2"'] \\ F(X_1) \arrow[r, "F(f)"] & F(X_2) \end{tikzcd}$$

(b) A *final* F-*coalgebra* (X_f, c_f) is a F-coalgebra such that for every F-coalgebra (X, c) there exists an unique homomorphism $f : X \to X_f$ of F-coalgebras.

It is then a theorem that (a) The identity map of a F-coalgebra (X, c) is a homomorphism of F-coalgebras. (b) Compositions of homomorphisms of F-coalgebras are homomorphisms of F-coalgebras. (c) A final F-coalgebra is unique up to isomorphism. It is now also possible to define a bisimulation between coalgebras but this concept will not be defined in this paper because it will not be used directly.

3.3 Automata in terms of coalgebra

Below 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 can be viewed as a coalgebraic map together with the output function that determines the subset of marked states.

Now we recall from [31] partial automata as coalgebras of a special functor in the category of sets with functions as morphisms. Let A be an arbitrary set (usually finite and referred to as the set of inputs or events). The free monoid of words (strings) over A is denoted by A^*. The empty string will be denoted by ε.

A *partial automaton* is a pair $S = (S, \langle o, t\rangle)$, where S is a set of states, and a pair of functions $\langle o, t\rangle\ : S \to 2 \times (1 + S)^A$, consists of an output function $o\ :\ S \to 2$ and a transition function $S \to (1+S)^A$. 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 $s1$. The transition function t associates to each state s in S a function $t(s)\ :\ A \to (1 + S)$. The set $1 + S$ is the disjoint union of S and 1. 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 $s \xrightarrow{a} s_a$. This notation can be extended by induction to arbitrary strings in A^*. Assuming that $s \xrightarrow{w} s_w$ has been defined, define $s \xrightarrow{wa}$ iff $t(s_w)(a) \in S$, in which case $s_{wa} = t(s_w)(a)$, also denoted by $s \xrightarrow{wa} s_{wa}$. It is easy to see that partial automata are coalgebras of the set functor $F = 2 \times (1 + (.)\ )^A$.

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 $a \in A$:

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

in which case: $f(s)_a = f(s.1_a)$.

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

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. It is important to notice that the coalgebraic concept of subautomata corresponds to the notion of strict subautomaton in [8]. In the sequel we use always subautomata in the coalgebraic sense defined above, i.e. strict subautomata are meant.

A *simulation* 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 $s' \in S'$:

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

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 $s' \in S'$:

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

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. It is immediate from the definition of bisimulation that two states are bisimilar iff they can make the same transitions and they give rise to the same outputs:

Proposition 3.1. *For any partial automaton $S = (S, \langle o, t\rangle)$ and any $s, s' \in S$:*

$$s \sim s' \textit{ iff } \forall w \in A^* : \; s \xrightarrow{w} \Longleftrightarrow s' \xrightarrow{w}, \textit{ in which case } o(s_w) = o'(s'_w).$$

3.4 Final automaton of partial languages

In this subsection an automaton is defined which is the final automaton among all partial automata. For the remainder of the paper it is important that the automaton considered is final. This makes then available the theorems of coinduction and of proofs of equality of partial languages by existence of a bisimulation.

Below a partial automaton of partial languages is defined over an alphabet (input set) A, denoted by $\mathcal{L} = (\mathcal{L}, \langle o_{\mathcal{L}}, t_{\mathcal{L}}\rangle)$. More formally, $\mathcal{L} = \{\Phi : \; A^* \to (1+2) \mid dom(\Phi) = \{w \in A^* \mid \Phi(w) \in 2\} \neq \emptyset$ is prefix-closed$\}$. To each partial language Φ a pair $\langle V, W\rangle$ can be assigned: $W = dom(\Phi)$ and $V = \{w \in dom(\Phi) \mid \Phi(w) = 1 (\in 2)\}$. Conversely, to a pair $\langle V, W\rangle \in \mathcal{L}$, a function Φ can be assigned : $\Phi(w) = 1$ if $w \in V$, $\Phi(w) = 0$ if $w \in W$ and $w \notin V$, and $\Phi(w)$ is undefined if $w \notin W$. Therefore we can write :

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

Now we define the partial automaton of partial languages $(\mathcal{L}, \langle o_{\mathcal{L}}, t_{\mathcal{L}}\rangle)$, where the transition function $t_{\mathcal{L}} \; : \mathcal{L} \to (1+\mathcal{L})^A$ is defined using Brzozowski input derivatives and $o_{\mathcal{L}}$ is also defined below. 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 $L \xrightarrow{w} L_w$ iff L_w is defined (iff $w \in L^2$).

3.5 Induction and coinduction

Induction is taught to undergraduate students in courses of algebra. The student learns that all elements of a sequence indexed by the natural numbers satisfy a specified property if (1) the first element of the sequence satisfies it and (2) if element $n \in \mathbb{N}$ satisfies the property then so does element $n+1$. This can be put in a more abstract setting as the proper definition of a function from the natural numbers to a set corresponding to the property concerned, and illustrated by a commutative diagram. Most of mathematics students only remember the simple sufficient condition and not the abstract setting.

Coinduction is a dual concept to induction. Many people use induction without bearing in mind its abstract (categorical or universally algebraic) meaning. Coinduction in its full generality must be put into a general framework of universal coalgebra that uses the category theory. Finality of a coalgebra enables coinductive definitions and proofs in a similar way as initiality of an algebra enables definitions and proofs by induction. In order to make the paper more accessible to a reader not very familiar with category theory we have prefered to introduce the coinduction only in its special form: on final coalgebra of partial languages. It is the same as with mathematical induction that is by many people understood only on the initial algebra of natural numbers with the (unary algebraic) structure given by the successor operation: $\forall n \in N: \ succ(n) = n+1$. Here definitions of functions by induction correspond to giving the successor on functions, hence yielding recursive formulas. Proofs by induction correspond to the very well known two-steps procedure, which amounts to verify that a relation is a congruence relation with respect to the successor operation. This is possible because natural numbers with the successor operation is the initial algebra in the category of all unary algebras, i.e. there is a unique morphism from the initial algebra of natural numbers to any unary algebra.

Similarly, a definition by coinduction amounts to give the corresponding structure, here output and derivatives on operations to be defined, and a proof by coinduction consists in verifying the conditions of bisimulation relation. We believe that giving a general categorical definition of coinduction would go far beyond the scope of the paper, the purpose of the paper is primarily control of discrete-event systems with coalgebra. Coinduction has been well covered by the existing literature on universal coalgebra [32], [33].

Coinduction is used as a proof and definition principle throughout this paper. The use of coinduction is limited to final coalgebras. Behavior equivalence of two elements of final coalgebra means that these are equal. Also notice that the elements of final coalgebras are equal to their behaviors (the identity is the unique behavior homomorphism). This feature is sometimes paraphrased as 'being is doing', because these elements behave as they are.

Proofs by coinduction consist in constructing appropriate relations: for instance a proof of equality of two elements of a final coalgebra consists in finding a bisimulation relation that relates them. Definition by coinduction of an operation on elements of a final coalgebra consists in defining the same coalgebraic structure on the operation (for instance we define binary operations on partial languages by defining derivatives and output functions further in this paper). More details about coinduction and finality can be found in [32] or [31].

3.6 Final automata and coinduction

Most of the rest of this subsection is recalled from [31].

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

Proof. It follows from Proposition 3.1. Indeed, if $K \sim L$ then for any $w \in A^*: K \xrightarrow{w} \Leftrightarrow L \xrightarrow{w}$, i.e. $w \in K^2$ iff $w \in L^2$, in which case $o(K_w) = o'(K'_w)$, i.e. $w \in K^1$ iff $w \in L^1$. It follows that $K = L$. The converse implication is also (trivially) true. □

Theorem 3.3. *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 $s \sim s'$.*

Proof. For the existence part of the theorem, we define the homomorphism l by putting for $s \in S$:

$$dom(l(s)) = \{w \in A^* : \ s \xrightarrow{w}\}$$

and

$$l(s) = ((l(s))^1, (l(s))^2) = (\{w \in A^* \mid s \xrightarrow{w} \text{ and } s_w \downarrow\}, \{w \in A^* \mid s \xrightarrow{w}\}).$$

Uniqueness of l follows from the fact that for any two homomorphisms $l, l' : S \to \mathcal{L}$ the relation

$$R = \{\langle l(s), l'(s) \rangle \in \mathcal{L} \times \mathcal{L} \mid s \in S\}$$

is a bisimulation. Therefore $l = l'$ follows from theorem 3.2. The last statement is immediate from the definition of l and Proposition 3.1. $\square$

We adopt the notation from [30], page 9, easily extended from automata to partial automata, and denote the minimal (in size of the state set) representation of a partial language L by $\langle L \rangle$. Hence, $\langle L \rangle = (DL, \langle o_{\langle L \rangle}, t_{\langle L \rangle} \rangle)$ is a subautomaton of $\mathcal{L}$ generated by L. This means that $o_{\langle L \rangle}$ and $t_{\langle L \rangle}$ are uniquely determined by the corresponding structure of $\mathcal{L}$. The carrier set of this minimal representation of L is denoted by DL, where $DL = \{L_u \mid u \in L^2\}$. Let us call this set the set of derivatives of L. Inclusion of partial languages that corresponds to a simulation relation is meant componentwise. The prefix closure of an (ordinary) language L is denoted by $\bar{L}$. Some further notation from [31] is used, e.g. 'zero' (partial) language is denoted by 0, i.e. $0 = (\emptyset, \{\varepsilon\})$.

There is yet another important concept that will be needed in this paper. Namely, given an (ordinary) language L, the suffix closure of L is defined by $\text{suffix}(L) = \{s \in A^* \mid \exists u \in A^* \text{ with } us \in L\}$. For partial languages, the suffix closure is defined in the same way as the prefix closure, i.e. componentwise. There is the following relation between the transition structure of L and its suffix closure operator.

Observation 3.4. *For any (partial) language L:* $\text{suffix}(L) = \cup_{u \in L^2} L_u$.

Proof. It is immediate from the fact that $L_u = (\{s \in A^* \mid us \in L^1\},\ \{s \in A^* \mid us \in L^2\})$. $\square$

3.7 Weak transitions

Control with partial observations implies that the observed traces are different from those with complete observations. This motivates the concept of weak transitions.

In the following definition we introduce the notion of weak derivative (transition). Roughly speaking it disregards unobservable steps, which correspond to so called internal moves in the framework of process algebras [26]. Let $A = A_o \cup A_{uo}$ be a partition of A into observable events (A_o) and unobservable (A_{uo}) events with the natural projection $P : A^* \to A_o^*$. Recall that $P(a) = \varepsilon$ for any $a \in A_{uo}$, $P(a) = a$ for $a \in A_o$, and P is catenative.

Definition 3.5. *(Nondeterministic weak transitions.) For a state s in partial automaton $S = (S, \langle o, t \rangle)$ and $a \in A$ we put $s \overset{P(a)}{\Rightarrow} s'$ if there exists $u \in A^*$ such that $P(u) = P(a)$ and $s \overset{u}{\to} s' = s_u$. We denote in this case $s \overset{P(a)}{\Rightarrow} s_u$.*

Remark 3.6. *In accordance with this notation $s \overset{\varepsilon}{\Rightarrow} s'$ is an abbreviation for $\exists \tau \in A_{uo}^*$ such that $s \overset{\tau}{\to} s_\tau = s'$. For $a \in A_o$ our notation means that there exist $\tau, \tau' \in A_{uo}^*$ such that $s \overset{\tau a \tau'}{\to} s_{\tau a \tau'}$. This definition can be extended to strings (words in A^*) in the obvious way:*
$s \overset{P(w)}{\Rightarrow} s'$ iff $\exists u \in A^*\ :\ P(u) = P(w)$ and $s \overset{u}{\to} s_u$. Denote in this case $s \overset{P(w)}{\Rightarrow} s_u$.

There may exist two or more $u \in A^*$ satisfying the condition in the definition of weak transition. Hence, the weak transition structure introduced above is not deterministic. We introduce deterministic weak transitions in $\mathcal{L}$, which are defined as unions of nondeteministic weak transitions:

Definition 3.7. *(Deterministic weak transitions.) Define for $a \in A_o$: $L \overset{a}{\Rightarrow} L_{\hat{a}}$ if $L \overset{P(a)}{\Rightarrow}$ and $L_{\hat{a}} := \cup_{\{s \in L^2 \mid P(s) = a\}} L_s$.*

4 Modular control with full observations.

Let us consider the concurrent behavior of local subplants $G_1, \dots, G_n$. Assume that the local alphabets of these subplants, A_i, not necessarily pairwise disjoint are such that $A_i = A_{iu} \cup A_{ic}$. First we assume that $A_{iu} \cap A_j = A_i \cap A_{ju}\ \forall i, j \in \mathbb{Z}_\mathrm{n} = \{1, \dots, n\}$. At the end of the section this assumption that does not fit in particular applications will be discarded. This assumption originally introduced in [44] means that the events shared by two local subsystems must have the same control status for both controllers associated to these subsystems. Denote $A_c = \cup_{i=1}^n A_{ic}$ and $A_u = A \setminus A_c$. We then still have the disjoint union $A = A_c \cup A_u$ and $A_u = \cup_{i=1}^n A_{iu}$ due to the assumption that $A_{iu} \cap A_j = A_i \cap A_{ju}$.

Denote $A = \cup_{i=1}^n A_i$ the global alphabet and $P_i : A \to A_i$ 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.

Let us notice that

Proposition 4.1. *$A_{iu} \cap A_j = A_i \cap A_{ju}$ is equivalent to the following inclusions: $A_{iu} \cap A_j \subseteq A_{ju}$ and $A_{ju} \cap A_i \subseteq A_{iu}$.*

Proof. The inclusions clearly follows from the equality. Let the inclusions hold and let us show the equality: for $a \in A_{iu} \cap A_j$ we have $a \in A_{ju}$, but also $a \in A_i$, because $A_{iu} \subseteq A_i$. The other inclusion of the equality can be shown similarly. $\square$

In the rest of this section global control synthesis will be compared to the local (modular) control synthesis. By global control synthesis we mean the construction of global supervisor that acts on the global plant. The local control synthesis means that local supervisors act on the local plants (modules) and the resulting controlled system is the parallel composition of the supervised local plants. In terms of behaviors, i.e. partial languages, the global global control synthesis is represented by the closed-loop language $(\|_{i=1}^n\ K_i)/_{A_u}(\|_{i=1}^n\ L_i)$ using the binary operation $K/_{A_u}L$ defined by coinduction in appendix. Similarly, modular control synthesis yields in terms of behaviors the partial language $\|_{i=1}^n\ (K_i/_{A_{iu}}L_i)$. We are interested whether or when the closed-loop languages are preserved by parallel compositions of local components (plants), i.e. the following languages are equal:

$$\|_{i=1}^n\ (K_i/_{A_{iu}}L_i) = (\|_{i=1}^n\ K_i)/_{A_u}(\|_{i=1}^n\ L_i).$$

The following theorem gives an answer. For simplicity we assume $n = 2$, the extension of our results to general n being easy as discussed later. Notation $\mathcal{L}_i,\ i = 1, 2$ is reserved for the final automaton of partial languages over alphabets A_i , $i = 1, 2$, respectively.

Theorem 4.2. *(Modular synthesis equals global synthesis in case of modular control with complete observations) If $A_{2u} \cap A_1 = A_2 \cap A_{1u}$, then*

$$(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) = (K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2).$$

Proof. The coinduction proof principle is used, i.e. it is sufficient to show that

$$R = \{\langle (K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2), (K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2)\rangle \in \mathcal{L} \times \mathcal{L},\ K_i, L_i \in \mathcal{L}_i,\ i = 1, 2\}$$

is a bisimulation.

(i) From the corresponding coinductive definitions of the parallel and supervised products, $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2)\downarrow$ iff $(K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2)\downarrow$ iff ($L_1\downarrow$ and $L_2\downarrow$).

(ii) Let $a \in A$ such that $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) \xrightarrow{a}$. According to the coinductive definition of the synchronous product several cases must be distinguished. Consider first the case $a \in A_1 \cap A_2$. Then we have $(K_1/_{A_{1u}}L_1) \xrightarrow{a}$ and $(K_2/_{A_{2u}}L_2) \xrightarrow{a}$. According to the definition of supervised product, 4 subcases must be distinguished. If $K_1 \xrightarrow{a}$, $K_2 \xrightarrow{a}$, $L_1 \xrightarrow{a}$ and $L_2 \xrightarrow{a}$, then $(K_1 \parallel K_2) \xrightarrow{a}$ and $(L_1 \parallel L_2) \xrightarrow{a}$, i.e. $(K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2) \xrightarrow{a}$. In the second subcase we have $K_1 \xrightarrow{a}$, $L_1 \xrightarrow{a}$, $K_2 \not\xrightarrow{a}$, $L_2 \xrightarrow{a}$, and $a \in A_{2u}$. Hence $(L_1 \parallel L_2) \xrightarrow{a}$ and $a \in A_{2u} \subseteq A_u$, i.e. $(K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2) \xrightarrow{a}$. The subcase $K_1 \not\xrightarrow{a}$, $L_1 \xrightarrow{a}$, $K_2 \xrightarrow{a}$, $L_2 \xrightarrow{a}$, and $a \in A_{1u}$ is symmetric. The last subcase is $K_1 \not\xrightarrow{a}$, $L_1 \xrightarrow{a}$, $K_2 \not\xrightarrow{a}$, $L_2 \xrightarrow{a}$, and $a \in A_{1u} \cap A_{2u}$. Here again $(L_1 \parallel L_2) \xrightarrow{a}$ and $a \in A_u$, i.e. $(K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2) \xrightarrow{a}$. The second case is $a \in A_1 \setminus A_2$. Here we have only $(K_1/_{A_{1u}}L_1) \xrightarrow{a}$. There are two subcases: either $K_1 \xrightarrow{a}$ and $L_1 \xrightarrow{a}$, which imply $(K_1 \parallel K_2) \xrightarrow{a}$ and $(L_1 \parallel L_2) \xrightarrow{a}$, i.e. $(K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2) \xrightarrow{a}$, or $K_1 \not\xrightarrow{a}$, $L_1 \xrightarrow{a}$, and $a \in A_{1u}$. Since $A_{1u} \subseteq A_u$ and $(L_1 \parallel L_2) \xrightarrow{a}$, the conclusion in the second subcase is the same: $(K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2) \xrightarrow{a}$. Finally, the third case $a \in A_2 \setminus A_1$ is completely symmetric to the second and therefore omitted. In order to verify that the new pairs of languages after $a-$ transition are included in R, it is sufficient to notice that $0 \| L = L$ for any partial language L. Owing to this property it is true that R is a bisimulation relation.

(iii) Let $(K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2) \xrightarrow{a}$ for $a \in A$. Two cases must be distinguished according to the definition of supervised product. First, let $(K_1 \parallel K_2) \xrightarrow{a}$ and $(L_1 \parallel L_2) \xrightarrow{a}$. Several subcases are now treated separately. If $a \in A_1 \cap A_2$, then $K_1 \xrightarrow{a}$, $K_2 \xrightarrow{a}$, $L_1 \xrightarrow{a}$ and $L_2 \xrightarrow{a}$, then both $(K_1/_{A_{1u}}L_1) \xrightarrow{a}$ and $(K_2/_{A_{2u}}L_2) \xrightarrow{a}$, i.e. also $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) \xrightarrow{a}$. If $a \in A_1 \setminus A_2$, then $K_1 \xrightarrow{a}$ and $L_1 \xrightarrow{a}$, i.e. $(K_1/_{A_{1u}}L_1) \xrightarrow{a}$ and also $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) \xrightarrow{a}$. The subcase $a \in A_2 \setminus A_1$ is fully symmetric to the previous subcase. Now let us consider the second case: $(K_1 \parallel K_2) \not\xrightarrow{a}$, $(L_1 \parallel L_2) \xrightarrow{a}$, and $a \in A_u$. Recall that $A_u = A_{1u} \cup A_{2u}$. If it happens that $a \in A_{1u} \cap A_{2u} \subseteq A_1 \cap A_2$, then $L_1 \xrightarrow{a}$, $L_2 \xrightarrow{a}$ and clearly both $(K_1/_{A_{1u}}L_1) \xrightarrow{a}$ and $(K_2/_{A_{2u}}L_2) \xrightarrow{a}$, i.e also $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) \xrightarrow{a}$. Problematic cases occur when either $a \in A_{1u} \setminus A_{2u}$ or $a \in A_{2u} \setminus A_{1u}$ and $a \in A_1 \cap A_2$. But according to our assumption we have in the latter case $a \in A_{2u} \cap A_1 \subseteq A_{1u} \cap A_2 \subseteq A_{1u}$ and similarly in the former case we obtain $a \in A_{2u}$. Therefore $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) \xrightarrow{a}$. If $a \in A_1 \setminus A_2$ or $a \in A_2 \setminus A_1$, then $a \in A_u$ implies $a \in A_{1u}$ or $a \in A_{2u}$, respectively, i.e. again $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) \xrightarrow{a}$. It can be shown by checking once again all cases that the new pairs of languages after $a-$ transition are included in R (using $0 \| L = L$ for any partial language L). $\square$

We recall that closed-loop languages defined by (local and global) supervised product correspond to infimal controllable superlanguages. According to our knowledge the result of Theorem 4.2 is new. Preservation of supremal controllable sublanguages in a modular DES is considered below. Notice that the inclusion $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) \subseteq (K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2)$ holds even without our assumption $A_{iu} \cap A_j = A_i \cap A_{ju}$. However the opposite inclusion may fail if the condition is not satisfied as is illustrated by the following example.

Example 4.3. *Let* $A = \{a_1, a_2, c, u, u_1, u_2\}$, $A_1 = \{a_1, u_1, u, c\}$, $A_2 = \{a_2, u_2, u, c\}$, $A_u = \{u_1, u_2, u\}$, $A_{1u} = \{u_1, u\}$, *and* $A_{2u} = \{u_2\}$. *Consider the following local specification and* *plant languages:*

$$
\begin{tikzcd}
& K_1 \arrow[dl, "a_1"'] \arrow[dr, "c"] & \\
(K_1)_{a_1} \arrow[dr, "u"] & & (K_1)_c \\
& (K_1)_{a_1u} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& L_1 \arrow[dl, "a_1"'] \arrow[dr, "c"] & \\
(L_1)_{a_1} \arrow[dr, "u"] & & (L_1)_c \arrow[dl, "u_1"'] \\
& (L_1)_{a_1u} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& K_2 \arrow[dl, "a_2"'] \arrow[dr, "c"] & \\
(K_2)_{a_2} & & (K_2)_c \arrow[dl, "u_2"'] \\
& (K_2)_{cu_2} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& L_2 \arrow[dl, "a_2"'] \arrow[dr, "c"] & \\
(L_2)_{a_2} \arrow[dr, "u"] & & (L_2)_c \arrow[dl, "u_2"'] \\
& (L_2)_{cu_2} &
\end{tikzcd}
$$

The notation $U = (K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2)$ and $V = (K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2)$ is used. From the definitions of the parallel and supervised products it follows that

$$
\begin{tikzcd}
& & U \arrow[dl, "a_1"'] \arrow[d, "a_2"] \arrow[dr, "c"] & & \\
& U_{a_1} \arrow[dl, "a_2"'] & U_{a_2} \arrow[dl, "a_1"'] & U_c \arrow[dl, "u_1"'] \arrow[dr, "u_2"] & \\
U_{a_1a_2} & U_{a_2a_1} & U_{cu_1} \arrow[dr, "u_2"] & & U_{cu_2} \arrow[dl, "u_1"'] \\
& & & U_{cu_1u_2} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& & V \arrow[dl, "a_1"'] \arrow[d, "a_2"] \arrow[dr, "c"] & & \\
& V_{a_1} \arrow[dl, "a_2"'] & V_{a_2} \arrow[dl, "a_1"'] & V_c \arrow[dl, "u_1"'] \arrow[dr, "u_2"] & \\
V_{a_1a_2} \arrow[dr, "u"] & V_{a_2a_1} \arrow[d, "u"] & V_{cu_1} \arrow[dr, "u_2"] & & V_{cu_2} \arrow[dl, "u_1"'] \\
& V_{a_1a_2u} & & V_{cu_1u_2} &
\end{tikzcd}
$$

Notice that in this example $(K_1/_{A_{1u}}L_1) \parallel (K_2/_{A_{2u}}L_2) \not\supseteq (K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2)$, which is caused by the shared event $u \in A_1 \cap A_2$ with $u \in A_{1u} \setminus A_{2u}$.

In the rest of this section we study the question when the optimal solutions to supervisory control problems (i.e. supremal controllable sublanguages) are preserved by the parallel composition. This problem has been studied algebraically in [44]. The concept of mutual controllability ([44]) plays the key role.

Definition 4.4. *Given partial languages $L_i = (L_i^1, L_i^2), L_j = (L_j^1, L_j^2)$, L_i and L_j are said to be mutually controllable if*

$$L_i^2(A_{ju} \cap A_i) \cap P_i(P_j)^{-1}(L_j^2) \subseteq L_i^2, \text{ and}$$

$$L_j^2(A_{iu} \cap A_j) \cap P_j(P_i)^{-1}(L_i^2) \subseteq L_j^2.$$

Mutual controllability can be viewed as local controllability of a local plant L_i^2 with respect to shared uncontrollable events in $(A_{ju} \cap A_i)$ and the local view of the other module $(P_i(P_j)^{-1}(L_j^2))$ as the new plant. This condition is important for modular computation of global supremal controllable sublanguages, denoted by $K/_C^S L$ using the coinductive definition from appendix. Local supremal controllable sublanguages, i.e. supremal sublanguages of K_i with respect to L_i and A_{iu}, are denoted by $K_i/_C^S L_i,\ i = 1, \dots, n$, and defined by coinduction. For simplicity we assume that $n = 2$. The following problem is addressed: under which conditions are supremal controllable sublanguages preserved by parallel composition:

$$(K_1/_C^S L_1) \parallel (K_2/_C^S L_2) = (K_1 \parallel K_2)/_C^S(L_1 \parallel L_2)?$$

Theorem 4.5 below gives a coalgebraic version of the proof for commutativity of supremal controllable sublanguages with synchronous product, which is however only a part of the problem treated in [44], namely it ignores blocking issues. Note that the main contribution of this paper is the extension of this result to the case of partial observations: commutativity of supremal normal sublanguages with the synchronous product. We believe that the proof presented in this paper is simpler than that of [44]. We believe that as in the monolithic supervisory control, partial observations do not bring themselves additional difficulty to handle the blocking issue. The blocking issue requires a different approach and it is believed by the authors that most of the framework proposed in this paper can be carried over to the framework which handles blocking though appropriately modified.

i.e. that known results, e.g. those of [31], can be applied. This is why the blocking issues are not treated in this paper.

Theorem 4.5. *(Sufficiency for modular equals global control synthesis for the supremal controllable sublanguage.) If in the above setting $A_{2u} \cap A_1 = A_2 \cap A_{1u}$, and L_1 and L_2 are mutually controllable, then $(K_1/^S_C L_1) \parallel (K_2/^S_C L_2) = (K_1 \parallel K_2)/^S_C(L_1 \parallel L_2)$.*

Proof. We use the coinduction proof principle, i.e. it is sufficient to show that

$$R = \{\langle (K_1/^S_C L_1) \parallel (K_2/^S_C L_2), (K_1 \parallel K_2)/^S_C(L_1 \parallel L_2)\rangle \in \mathcal{L} \times \mathcal{L},\ K_1, K_2, L_1, L_2 \in \mathcal{L}\}$$

is a bisimulation.

(i) From the corresponding coinductive definitions of the parallel product and supremal controllable sublanguage, $(K_1/^S_C L_1) \parallel (K_2/^S_C L_2) \downarrow$ iff $(K_1 \parallel K_2)/^S_C(L_1 \parallel L_2) \downarrow$.

(ii) Let $a \in A$ such that $(K_1/^S_C L_1) \parallel (K_2/^S_C L_2) \xrightarrow{a}$. According to the coinductive definition of the synchronous product several cases must be distinguished. Consider first the case $a \in A_1 \cap A_2$. Then we have $(K_1/^S_C L_1) \xrightarrow{a}$ as well as $(K_2/^S_C L_2) \xrightarrow{a}$. According to the coinductive definition of the supremal controllable sublanguage $K_1 \xrightarrow{a}$, $L_1 \xrightarrow{a}$, $K_2 \xrightarrow{a}$, $L_2 \xrightarrow{a}$ and for $i = 1, 2$ and $u \in A^*_{iu} :\ (L_i)_a \xrightarrow{u} \Rightarrow\ (K_i)_a \xrightarrow{u}$. Therefore $(K_1 \parallel K_2) \xrightarrow{a}$ as well as $(L_1 \parallel L_2) \xrightarrow{a}$. It remains to show that $\forall u \in A^*_u :\ (L_1 \parallel L_2)_a \xrightarrow{u} \Rightarrow\ (K_1 \parallel K_2)_a \xrightarrow{u}$. It follows from the fact that according to the coinductive definition of the synchronous product inductively applied there exist $v_1 \in A^*_{1u}$ and $v_2 \in A^*_{2u}$ such that $(L_1 \parallel L_2)_{au} = (L_1)_{av_1} \parallel (L_2)_{av_2}$, where $v_1 \in A^*_{1u}$ and $v_2 \in A^*_{2u}$. Indeed, in fact $v_1 = P_1(u)$ and $v_2 = P_2(u)$. It follows that $(K_i)_a \xrightarrow{v_i}$ for i=1,2. Hence also $(K_1 \parallel K_2)_a \xrightarrow{u} (K_1)_{av_1} \parallel (K_2)_{av_2}$ according to the coinductive definition of the synchronous product inductively applied. Consider now the case $a \in A_1 \setminus A_2$. Then there must be $(K_1/^S_C L_1) \xrightarrow{a}$. This means that $K_1 \xrightarrow{a}$, $L_1 \xrightarrow{a}$, and $\forall u \in A^*_{1u} :\ (L_1)_a \xrightarrow{u} \Rightarrow\ (K_1)_a \xrightarrow{u}$. We conclude that $(K_1 \parallel K_2) \xrightarrow{a}$ as well as $(L_1 \parallel L_2) \xrightarrow{a}$. It remains to show that $\forall u \in A^*_u :\ (L_1 \parallel L_2)_a \xrightarrow{u} \Rightarrow\ (K_1 \parallel K_2)_a \xrightarrow{u}$. Let $(L_1 \parallel L_2)_a \xrightarrow{u}$ for some $u \in A^*_u$. There exist $u_i \in A^*_{iu}$ for $i = 1, 2$: namely $u_i = P_i(u)$ such that $(L_1 \parallel L_2)_a \xrightarrow{u} (L_1)_{au_1} \parallel (L_2)_{u_2}$, because $a \notin A_2$. Since $(L_1)_a \xrightarrow{u_1}$, it follows from above that $(K_1)_a \xrightarrow{u_1}$. However, we must still show that $K_2 \xrightarrow{u_2}$. Notice that from the controllability of $K_2/^S_C L_2$ with respect to L_2 and A_{2u} and $L_2 \xrightarrow{u_2}$ we have $(K_2/^S_C L_2) \xrightarrow{u_2}$, i.e. in particular $K_2 \xrightarrow{u_2}$. Hence, $(K_1 \parallel K_2)_a \xrightarrow{u} (K_1)_{au_1} \parallel (K_2)_{u_2}$, which means that $(K_1 \parallel K_2)/^S_C(L_1 \parallel L_2) \xrightarrow{a}$. The remaining case $a \in A_2 \setminus A_1$ is symmetric to the previous one.

(iii) Let $(K_1 \parallel K_2)/^S_C(L_1 \parallel L_2) \xrightarrow{a}$ for $a \in A$. Thus, according to the coinductive definition of the supremal controllable sublanguage we have $(K_1 \parallel K_2) \xrightarrow{a}$, $(L_1 \parallel L_2) \xrightarrow{a}$, and $\forall u \in A^*_u$:

$(L_1 \parallel L_2)_a \xrightarrow{u} \Rightarrow \ (K_1 \parallel K_2)_a \xrightarrow{u}$. Different cases as in (ii) must be distinguished. First, let $a \in A_1 \cap A_2$. Then $K_1 \xrightarrow{a}$, $L_1 \xrightarrow{a}$, $K_2 \xrightarrow{a}$, $L_2 \xrightarrow{a}$. In order to prove that $(K_1/^S_C L_1) \parallel (K_2/^S_C L_2) \xrightarrow{a}$ it remains to show that for $i = 1, 2$ and $u \in A^*_{iu}$: $(L_i)_a \xrightarrow{u} \Rightarrow \ (K_i)_a \xrightarrow{u}$. Since A_1 and A_2 are in general overlapping, we have in general $u \in (A_{1u} \cup A_{2u})^* = A^*_u$. We must show that $(L_1 \parallel L_2)_a = (L_1)_a \parallel (L_2)_a \xrightarrow{u}$. First, let $u \in A^*_{1u}$ be such that $(L_1)_a \xrightarrow{u}$. Let us prove that $(L_1 \parallel L_2)_a = ((L_1)_a \parallel (L_2)_a) \xrightarrow{u} (L_1)_{au} \parallel (L_2)_{au_2}$, where $u_2 = P_2(u)$. It must be shown that $(L_2)_a \xrightarrow{u_2}$, i.e. $au_2 \in L^2_2$. Since $a \in L^2_2$, $au \in L^2_1$, $P_1(au) = au$, $P_2(au) = au_2$, and by our initial assumption $u_2 \in A_{2u} \cap A_1 = A_2 \cap A_{1u}$, the string $au_2 \in L^2_2(A_{1u} \cap A_2)^* \cap P_2(P_1)^{-1}(L^2_1)$, we deduce $au_2 \in L^2_2$ using the mutual controllability condition. Hence, $(L_2)_a \xrightarrow{u_2}$, and $(L_1 \parallel L_2)_a \xrightarrow{u}$ $(L_1)_{au} \parallel (L_2)_{au_2}$. Recall from above that $\forall u \in A^*_u$: $(L_1 \parallel L_2)_a \xrightarrow{u} \Rightarrow \ (K_1 \parallel K_2)_a \xrightarrow{u}$. Therefore we have $(K_1 \parallel K_2)_a \xrightarrow{u} (K_1)_{au} \parallel (K_2)_{au_2}$. This means in particular that $(K_1)_a \xrightarrow{u}$. In the symmetric way it can be shown using the mutual controllability condition that for any $u \in A^*_{2u}$: $(L_2)_a \xrightarrow{u} \Rightarrow \ (K_2)_a \xrightarrow{u}$. Let us consider the case $a \in A_1 \setminus A_2$. Then $K_1 \xrightarrow{a}$, and $L_1 \xrightarrow{a}$. In order to prove that $(K_1/^S_C L_1) \parallel (K_2/^S_C L_2) \xrightarrow{a}$ it must be shown that $(K_1/^S_C L_1) \xrightarrow{a}$. It remains to show that for all $u \in A^*_{1u}$: $(L_1)_a \xrightarrow{u} \Rightarrow \ (K_1)_a \xrightarrow{u}$. Let $(L_1)_a \xrightarrow{u}$ for a $u \in A^*_{1u}$. We show first that $(L_1 \parallel L_2)_a \xrightarrow{u} (L_1)_{au} \parallel (L_2)_{u_2}$, where $u_2 = P_2(u)$. In order to see that $L_2 \xrightarrow{u_2}$, mutual controllability is applied: $u_2 = \varepsilon u_2 \in L^2_2(A_{1u} \cap A_2)^* \cap P_2(P_1)^{-1}(L^2_1) \subseteq L^2_2$, because $u_2 \in (A_{1u} \cap A_2)^*$, $P_1(au) = au$, $P_2(au) = u_2$, and $au \in L^2_1$. Therefore $(K_1 \parallel K_2)_a \xrightarrow{u}$ $(K_1)_{au} \parallel (K_2)_{u_2}$. This means in particular that $(K_1)_a \xrightarrow{u}$. The conclusion is $(K_1/^S_C L_1) \xrightarrow{a}$, i.e. also $(K_1/^S_C L_1) \parallel (K_2/^S_C L_2) \xrightarrow{a}$. The remaining case $a \in A_2 \setminus A_1$ is again symmetric to the previous one.

□

The following question naturally appears: do supervised product and parallel product commute under more general structural conditions? We have already pointed out that the inclusion $(K_1/_{A_{1u}} L_1) \parallel (K_2/_{A_{2u}} L_2) \subseteq (K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2)$ always holds true for $A_u = A_{1u} \cup A_{2u}$. Some special cases, where the condition $A_{2u} \cap A_1 = A_2 \cap A_{1u}$ does not hold might still be of interest. For instance, in the DES model of the IEEE 802.11 protocol for wireless local area networks the condition $A_{1c} \cap A_{2c} = \emptyset$ holds instead. In the case $A_{1c} \cap A_{2c} = \emptyset$ we have also $A_u = A_{1u} \cup A_{2u}$, i.e. in particular $A_{iu} \subseteq A_u$ for $i = 1, 2$. Therefore we have:

Corollary 4.6. *If $A_{1c} \cap A_{2c} = \emptyset$, then*

$$(K_1/_{A_{1u}} L_1) \parallel (K_2/_{A_{2u}} L_2) \subseteq (K_1 \parallel K_2)/_{A_u}(L_1 \parallel L_2).$$

This means that the synchronized local control synthesis gives a smaller language than the global control synthesis. Roughly speaking, it can be useful when the safety is the main issue: local control synthesis is safe. However, the language achieved in the modular synthesis is in general smaller and the equality in Corollary 4.6 does not hold. The same inclusion as in Corollary 4.6 holds for the supremal controllable sublanguage: although modular (local) synthesis is safe (under shared event assumption), it is not in general optimal!

5 Modular control with partial observations

In this section we assume that each module G_i has only partial observation of its events, i.e. $A_i = A_{o,i} \cup A_{uo,i}$ is the decomposition of local events into locally observable and locally unobservable. The global system has observation set $A_o = \cup^n_{i=1} A_{o,i} \subseteq A = \cup^n_{i=1} A_i$. Some additional notation is needed to set up our framework. 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 $P_i : A^* \to A_i^*,\ i = 1, 2$. Partial observations in individual modules are expressed via local projections $P_i^{loc} : A_i^* \to A_{o,i}^*$, while global projection is denoted by $P : A^* \to A_o^*$. Local plant languages will be denoted by L_i, $i \in Z_n = \{1, \ldots, n\}$ and local specification languages by K_i, $i \in Z_n$. We assume from now on that $n = 2$ and that the global plant L and specification K languages are decomposable into local plant and local specification languages: $L = L_1 \parallel L_2$ and $K = K_1 \parallel K_2$. Note that this concept of decomposability is a special instance of decomposability studied in decentralized control and is also called separability, cf. [43].

Similarly as for completely observed modular DES we assume that the local modules agree on the observational status of the shared events, i.e. that $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$.

It is easy to show that this concept of decomposability corresponds exactly to the decomposability from e.g. [28]:

Definition 5.1. *We say that $L \subseteq A^*$ is decomposable with respect to P_1 and P_2 if $L = P_1^{-1}P_1(L) \cap P_2^{-1}P_2(L)$.*

Indeed, we have as a special case of Lemma 1 in [15].

Proposition 5.2. *$L \subseteq A^*$ is decomposable with respect to P_1 and P_2 iff there exists $L_1 \subseteq A_1^*$ and $L_2 \subseteq A_2^*$ such that $L = L_1 \parallel L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$.*

A pioneer study in modular supervisory control with partial observations has been done in [28], where special cases that occur in broadcast networks are studied. The setting and problems studied in this paper are more general. In order to study closed-loop languages of supervisory control we need the following auxiliary concept. It reflects the fact that due to partial observations it is not possible to distinguish between states:

Definition 5.3. *(Observational indistinguishability relation on S.) A binary relation $Aux(S)$ on S, called the* observational indistinguishability relation *is the smallest relation satisfying:*

(i) $\langle s_0, s_0 \rangle \in Aux(S)$

(ii) *If $\langle s, t \rangle \in Aux(S)$ then $\forall a \in A$: $(s \overset{P(a)}{\Rightarrow} s'$ for some s' and $t \overset{P(a)}{\Rightarrow} t'$ for some t' $) \Rightarrow \langle s', t' \rangle \in Aux(S)$*

From the definition of weak transitions it follows that (ii) is equivalent to (ii)' and (iii)' below:
(ii)' If $\langle s, t \rangle \in Aux(S)$ then : $(s \overset{\varepsilon}{\Rightarrow} s'$ for some s' and $t \overset{\varepsilon}{\Rightarrow} t'$ for some t' $) \Rightarrow \langle s', t' \rangle \in Aux(S)$
(iii)' If $\langle s, t \rangle \in Aux(S)$ then $\forall a \in A_o$: $(s \xrightarrow{a} s_a$ and $t \xrightarrow{a} t_a$ $) \Rightarrow \langle s_a, t_a \rangle \in Aux(S)$.
The notation $\lfloor s \rfloor_{Aux(S)} = \{s' \in S : \langle s, s' \rangle \in Aux(S)\}$ from [20] is useful. $Aux(S_1)$ can be characterized by the following lemma.

Lemma 5.4. *For any $s, s' \in S$: $\langle s, s' \rangle \in Aux(S_1)$ iff there exist two strings $w, w' \in K^2$ such that $P(w) = P(w')$, $s = (s_0)_w$ and $s' = (s_0)_{w'}$.*

In order to define the partial observation countrepart of $(K/_{A_u}L)$ (supervised product with partial observations), we need auxiliary relations $Aux(S)$ (introduced in definition 5.3) for the special case of minimal partial automaton $S = \langle K \rangle$. We will write $Aux(K)$ instead of $Aux(\langle K \rangle)$. Notice that it is possible to extend the definition of $Aux(S)$ to $Aux(Pwr(S))$ with the only difference, that the propagation of this relation is realized by deterministic weak transitions introduced in definition 3.7. In the case of the final automaton of partial languages the same construction of extended observational indistinguishability relation is to be realized on $\mathrm{Pwr}(\mathrm{suffix}(K))$. Now we prepare the coinductive definition of the supervised product with partial observations. This definition will consider arguments from $\mathrm{Pwr}(\mathrm{suffix}(K))$ and $\mathrm{Pwr}(\mathrm{suffix}(L))$ rather than from DK and DL. According to Observation 3.4 we will work with unions of the form $\cup_{i=1}^{k} K_{s_i} \in \mathrm{Pwr}(\mathrm{suffix}(K))$, where $P(s_1) = \cdots = P(s_k)$. In order to keep the notation simple, we denote the extension of $Aux(K)$ to such unions of derivatives by $Aux(K)$.

Now we give a formal definition of $Aux(K)$ extended to $\mathrm{Pwr}(\mathrm{suffix}(K))$.

Definition 5.5. *(Extension of $Aux(K)$ from DK to Pwr(suffix(K))). A binary relation $Aux(K) \subseteq (\mathrm{Pwr}(\mathrm{suffix}(K)))^2$, called* observational indistinguishability relation *is the smallest relation satisfying:*

(i) $\langle(K, K) \in Aux(K)$

(ii) *If* $\langle M, N\rangle \in Aux(K)$ *then* $\forall a \in A:\ M \xrightarrow{a} M_a$ *and* $N \xrightarrow{a} N_a \ \Rightarrow\ \langle M_a, N_a\rangle \in Aux(K)$

(iii) *If* $\langle M, N\rangle \in Aux(K)$ *then* $\forall m, n \in Z_+$*: if* $M \stackrel{\varepsilon}{\Rightarrow} M_1, M \stackrel{\varepsilon}{\Rightarrow} M_2, \ldots, M \stackrel{\varepsilon}{\Rightarrow} M_n$*, and* $N \stackrel{\varepsilon}{\Rightarrow} N_1, \ldots, N \stackrel{\varepsilon}{\Rightarrow} N_m$*, then* $\langle \cup_{i=1}^{n} M_i, \cup_{j=1}^{m} N_j\rangle \in Aux(K)$.

Clearly, a natural extension of Lemma 5.4 holds. Namely, $\langle \cup_{i=1}^{k} K_{s_i}, \cup_{j=1}^{l} L_{t_j}\rangle \in Aux(K)$, where $P(s_1) = \cdots = P(s_k)$ and $P(t_1) = \cdots = P(t_l)$ iff $P(s_1) = P(t_1)$, which implies naturally $P(s_i) = P(t_j)\ \forall i, j$. The notation $\cup_{i=1}^{k} K_{s_i} \approx_{Aux}^{K} \cup_{j=1}^{l} L_{t_j}$ is also used.

Definition 5.6. *(*Supervised product under partial observations.*) Define the following binary operation on (partial) languages called supervised product under partial observations for all* $M \in \mathrm{Pwr}(\mathrm{suffix}(K))$ *and* $N \in \mathrm{Pwr}(\mathrm{suffix}(L))$*:*

$$(M/_{A_u}^{A_o} N)_a =$$

(1) $M_a/_{A_u}^{A_o} N_a$ *if* $M \xrightarrow{a}$ *and* $N \xrightarrow{a}$*;*

(2) $(\cup_{\{M':\langle M', M\rangle \in Aux(K)\}} M'_a)/_{A_u}^{A_o} N_a$ *if* $M \not\xrightarrow{a}$ *and* $\exists M' \in DK : M' \approx_{Aux}^{K} M$ *such that* $M' \xrightarrow{a}$ *and* $N \xrightarrow{a}$ *and* $a \in A_c \cup A_o$*;*

(3) $0/_{A_u}^{A_o} N_a$ *if* $M \not\xrightarrow{a}$ *and* $(\forall M' \in DK\ : M' \approx_{Aux}^{K} M)\ M' \not\xrightarrow{a}$ *and* $N \xrightarrow{a}$ *and* $a \in A_u \cap A_o$*;*

(4) $M/_{A_u}^{A_o} N_a$ *if* $M \not\xrightarrow{a}$ *and* $N \xrightarrow{a}$ *and* $a \in A_u \cap A_{uo}$*;*

(5) $\emptyset$ *otherwise*

and $(M/_{A_u}^{A_o} N)\downarrow$ *iff* $N\downarrow$.

Remark 5.7. *We consider from now on an order relation on partial languages induced by their second components only, i.e. we write* $K \subseteq L$ *iff* $K^2 \subseteq L^2$*. The same applies for infimum and supremum operations. Note that only the second condition (ii) of simulation relations must be checked to prove such defined inclusion of partial languages.*

Let us recall from [20] that

Theorem 5.8. $(K/_{A_u}^{A_o} L) = \inf\{M \supseteq K :\ M$ *is controllable with respect to* L *and* A_{uc} *and observable with respect to* L *and* $P\}$. $(K/_{U}^{O} L)$ *equals the infimal controllable and observable superlanguage of* K.

Problem 5.9. *(Modular control synthesis equals global control synthesis for modular DES with partial observations) The local supervised products (under partial observations) are denoted by* $K_i/^{A_{o,i}}_{A_{iu}}L_i$, $i = 1,2$ *and the global supervised product (under partial observations) by* $K/^{A_o}_{A_u}L$. *We are interested, whether/when it is true that*

$$(K_1/^{A_{o,1}}_{A_{1u}}L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}}L_2) = (K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2).$$

The following lemmas are needed.

Lemma 5.10. *If* $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$ *then for any* $s, s' \in K = K_1 \parallel K_2$ *we have: if* $P(s) = P(s')$ *then for* $i = 1,2$*:* $P_i^{loc}P_i(s) = P_i^{loc}P_i(s')$.

Proof. The claim will be proven by structural induction on $P(s) = P(s') \in A_o^*$. For $P(s) = P(s') = \varepsilon$ it is easy to see that for $i = 1,2$: $P_i^{loc}P_i(s) = \varepsilon = P_i^{loc}P_i(s')$, because $s, s' \in A_{uo}^*$ implies that $P_i(s), P_i(s') \in A_{uo,i}^*$, whence the above equality holds. The induction step follows. Let for $P(s) = P(s') = w \in A_o^*$ the implication holds true and let us show that for $s, s' :\ P(s) = P(s') = wa \in A_o^*$, $P(s) = P(s')$ implies for $i = 1,2$: $P_i^{loc}P_i(s) = P_i^{loc}P_i(s')$. It should be clear that s and s' are of the following form: $s = t\tau a\tau'$ and $s' = t'\sigma a\sigma'$, where $\tau, \tau', \sigma, \sigma' \in A_{uo}^*$. For $i = 1,2$: $P_i^{loc}P_i(s) = P_i^{loc}P_i(t\tau)P_i^{loc}P_i(a\tau')$ and $P_i^{loc}P_i(s') = P_i^{loc}P_i(t'\sigma)P_i^{loc}P_i(a\sigma')$, because P is catenative. Notice that $P(t\tau) = P(t'\sigma) = w$, i.e. according to the induction hypothesis $P_i^{loc}P_i(t\tau) = P_i^{loc}P_i(t'\sigma)$ and it is sufficient to show that: $P_i^{loc}P_i(a\tau') = P_i^{loc}P_i(a\sigma')$, which amounts to show that $P_i^{loc}P_i(\tau') = P_i^{loc}P_i(\sigma')$. Different cases must be considered. If both $\tau', \sigma' \in A_1 \cap A_2$, then it follows from our assumption that $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$ that both $\tau', \sigma' \in A_{uo,1} \cap A_{uo,2}$, i.e. $P_i^{loc}P_i(\tau') = P_i^{loc}P_i(\sigma') = \varepsilon$ for $i = 1,2$. If $\tau' \in A_1 \setminus A_2$ and $\sigma' \in A_2 \setminus A_1$, then it must be that $\tau' \in A_{uo,1}$, because $\tau' \in A_{uo}$. Therefore $P_1^{loc}P_1(\tau') = P_1^{loc}P_1(\sigma') = \varepsilon$ and similarly $P_2^{loc}P_2(\tau') = P_2^{loc}P_2(\sigma') = \varepsilon$. Other cases can be treated similarly. For instance, if $\tau' \in A_1 \cap A_2$ and $\sigma' \in A_1 \setminus A_2$, then we have $\tau' \in A_{uo,1} \cap A_{uo,2}$ and $\sigma' \in A_{uo,1}$, hence $P_1^{loc}P_1(\tau') = P_1^{loc}P_1(\sigma') = \varepsilon$ and $P_2^{loc}P_2(\tau') = P_2^{loc}P_2(\sigma') = \varepsilon$. □

Lemma 5.11. *If* $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$ *then* $PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc} = PP_1^{-1}$ *and* $PP_2^{-1}(P_2^{loc})^{-1}P_2^{loc} = PP_2^{-1}$.

Remark 5.12. *Remark that the claim of the lemma is not trivial, because unlike* $P_1^{loc}(P_1^{loc})^{-1} = I$*, in general* $(P_1^{loc})^{-1}P_1^{loc} \neq I$*,* I *being the identity function.*

Proof. We prove the first statement of the lemma by structural induction on string $s \in A_1^*$ (the other equality being symmetric). For $s = \varepsilon$ clearly $PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(\varepsilon) = PP_1^{-1}(A_{uo,1}^*) = P((A_2 \setminus A_1)^*)$, because of our assumption that $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$ and $A_{uo,1} \subseteq A_{uo}$. But we have also $PP_1^{-1}(\varepsilon) = P((A_2 \setminus A_1)^*)$, i.e. for $s = \varepsilon$, $PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(s) = PP_1^{-1}(s)$.
The induction step follows: let $PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(s) = PP_1^{-1}(s)$. Then $PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(sa) = PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(s)PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(a) = PP_1^{-1}(s)PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(a)$ using the induction hypothesis. Therefore it is sufficient to show that for $a \in A_1^*$: $PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(a) = PP_1^{-1}(a)$. Notice that for $a \in A_{uo,1} \subseteq A_{uo}$ we have $PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(a) = PP_1^{-1}(P_1^{loc})^{-1}(\varepsilon) = P((A_2 \setminus A_1)^*)$ as it has already been computed above. On the other hand, $PP_1^{-1}(a) = P((A_2 \setminus A_1)^* a (A_2 \setminus A_1)^*) = P((A_2 \setminus A_1)^*)$ by taking into account the facts that $P(a) = \varepsilon$, P is catenative, and the star operation is idempotent with respect to concatenation. For $a \in A_{o,1}$ we obtain: $PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(a) = PP_1^{-1}(P_1^{loc})^{-1}(a) = PP_1^{-1}(A_{uo,1}^* a A_{uo,1}^*) = P((A_2 \setminus A_1)^* a (A_2 \setminus A_1)^*) = PP_1^{-1}(a)$ using the same arguments as above: especially $A_{uo,1} \subseteq A_{uo}$, which means that $P(A_{uo,1}^*) = \varepsilon$. □

We have the following result concerning the infimal controllable and observable superlanguages.

Theorem 5.13. *(Modular equals global control synthesis for closed-loop languages under partial observations) If $A_c \subseteq A_o$, $A_{2u} \cap A_1 = A_2 \cap A_{1u}$, and $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$ then*

$$(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2) = (K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2).$$

Remark 5.14. *Notice that the condition $A_c \subseteq A_o$ means that global observability is equivalent to global normality. As a consequence globally optimal solutions to supervisory control problems always exist.*

Proof. The coinductive proof principle is used, i.e. we will show that the relation below is a bisimulation.

$$R = \{\langle [(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2)]_s, [(K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2)]_s \rangle \mid$$
$$s \in [(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2)]^2\}.$$

(i) It is clear from the coinductive definitions of the synchronized and supervised products that both sides have the same logical outputs.

(ii) Let $[(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2)]_s \xrightarrow{a}$ for $a \in A$. According to the coinductive definition of parallel product $[(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2)]_s = [(K_1/^{A_{o,1}}_{A_{1u}} L_1)]_{s_1} \parallel [(K_2/^{A_{o,2}}_{A_{2u}} L_2)]_{s_2}$ with $P_1(s) = s_1$ and $P_2(s) = s_2$. Using the definition of parallel product three cases must be distinguished.
(A): $a \in A_1 \cap A_2$. Then $[(K_1/^{A_{o,1}}_{A_{1u}} L_1)]_{s_1} \xrightarrow{a}$ and $[(K_2/^{A_{o,2}}_{A_{2u}} L_2)]_{s_2} \xrightarrow{a}$. According to the coinductive definition of the supervised product with partial observations repeatedly applied the three cases below can occur for both $i = 1, 2$:

$$[(K_i/^{A_{o,i}}_{A_{iu}} L_i)]_{s_i a} = \begin{cases} (1) & (K_i)_{s_i a}/^{A_{o,i}}_{A_{iu}}(L_i)_{sa} \\ (2) & [\cup_{\{s_i' :\, P_i^{loc}(s_i') = P_i^{loc}(s_i)\}} (K_i)_{s_i' a}]/^{A_{o,i}}_{A_{iu}}(L_i)_{s_i a} \\ (3) & 0/^{A_{o,i}}_{A_{iu}}(L_i)_{s_i a} \text{ and } a \in A_u \end{cases}$$

In case that for both $i = 1$ and $i = 2$ case (1) occurs, the situation is easy: $K_1 \xrightarrow{s_1 a}$, $L_1 \xrightarrow{s_1 a}$, $K_2 \xrightarrow{s_2 a}$, $L_2 \xrightarrow{s_2 a}$, i.e. $(K_1 \parallel K_2)_s \xrightarrow{a}$ and $(L_1 \parallel L_2)_s \xrightarrow{a}$. Therefore $[(K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2)]_s \xrightarrow{a}$. The most difficult is the situation when for both $i = 1$ and $i = 2$ case (2) occurs. Situations, when for $i = 1$ case (1) occurs and for $i = 2$ case (2) occurs or for $i = 1$ case (2) occurs and for $i = 2$ case (1) occurs, can be covered by the situation where for both i case (2) occurs. It must be shown that from ($\exists s_1' \in A_1^*$ and $s_2'' \in A_2^*$ such that $s_1' a \in K_1^2$, $s_2'' a \in K_2^2$, $P_1^{loc}(s_1') = P_1^{loc}(s_1)$, and $P_2^{loc}(s_2'') = P_2^{loc}(s_2)$) follows that there exists $s' \in A^*$ such that $P(s') = P(s)$ and $s'a \in K^2 = (K_1 \parallel K_2)^2$. Then we will have $[(K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2)]_s \xrightarrow{a}$, because we have always $(L_1 \parallel L_2)_s \xrightarrow{a}$. We obtain: $P_1^{loc} P_1(sa) \in P_1^{loc}(K_1^2)$ and similarly, $P_2^{loc} P_2(sa) \in P_2^{loc}(K_2^2)$. Thus, $sa \in P_1^{-1}(P_1^{loc})^{-1} P_1^{loc}(K_1^2) \cap P_2^{-1}(P_2^{loc})^{-1} P_2^{loc}(K_2^2)$, i.e. $P(sa) \in PP_1^{-1}(P_1^{loc})^{-1} P_1^{loc}(K_1^2) \cap PP_2^{-1}(P_2^{loc})^{-1} P_2^{loc}(K_2^2)$ We need to show that $P(sa) \in P(K^2) = PP_1^{-1}(K_1^2) \cap PP_2^{-1}(K_2^2)$. It follows from Lemma 5.11 that for $i = 1, 2$: $PP_i^{-1}(P_i^{loc})^{-1} P_i^{loc}(K_i) = PP_i^{-1}(K_i)$, hence $P(sa) \in P(K^2)$, i.e. $\exists t \in A^*$ such that $P(t) = P(sa)$ and $t \in K^2 = (K_1 \parallel K_2)^2$. Since $a \in A_c \subseteq A_o$ (the case $a \in A_u$ being trivial), we have also $\exists s' \in A^*$ such that $P(s) = P(s')$ and

$s'a \in K^2$. The remaining cases are when for $i = 1$ and/or $i = 2$ case (3) occurs. But then $a \in A_u$. Therefore $[(L_1 \parallel L_2)]_s \xrightarrow{a}$ implies $[(K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2)]_s \xrightarrow{a}$ according to the definition of supervised product.

Consider now the case (B): $a \in A_1 \setminus A_2$. This case is easier. Indeed, $[(K_1/^{A_{o,1}}_{A_{1u}} L_1)]_{s_1} \parallel [(K_2/^{A_{o,2}}_{A_{2u}} L_2)]_{s_2} \xrightarrow{a}$ means that $[(K_1/^{A_{o,1}}_{A_{1u}} L_1)]_{s_1} \xrightarrow{a}$. We have three possibilities according to the definition of supervised product. The first case is $(K_1)_{s_1} \xrightarrow{a}$. This implies $[(K_1 \parallel K_2)]_s = \{(K_1)_{s_1} \parallel (K_2)_{s_2}\} \xrightarrow{a}$. The second possibility is $\exists s_1' \in K_1^*$ such that $P_1^{loc}(s_1') = P_1^{loc}(s_1)$ and $(K_1)_{s_1'} \xrightarrow{a}$. Thus, we have $s_1'a \in K_1^2$, i.e. Since $[K_2/^{A_{o,2}}_{A_{2u}} L_2] \xrightarrow{s_2}$, we deduce that there exists $s_2' \in (K_2)^2$ such that $P_2^{loc}(s_2') = P_2^{loc}(s_2)$. Otherwise $a \in A_u$, which case is easy. Thus, $P_2^{loc}P_2(s) \in (P_2^{loc}(K_2))^2$. Since $a \notin A_2$: $P_2^{loc}P_2(sa) = P_2^{loc}P_2(s) \in (P_2^{loc}(K_2))^2$. The continuation is now the same as in the case $a \in A_1 \cap A_2$: $sa \in P_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(K_1) \cap P_2^{-1}(P_2^{loc})^{-1}P_2^{loc}(K_2)$, i.e. $P(sa) \in PP_1^{-1}(P_1^{loc})^{-1}P_1^{loc}(K_1^2) \cap PP_2^{-1}(P_2^{loc})^{-1}P_2^{loc}(K_2^2) = P(K^2)$. The last possibility is $a \in A_{1u} \subseteq A_u$, which implies together with $[(L_1 \parallel L_2)]_s \xrightarrow{a}$ that $[(K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2)]_s \xrightarrow{a}$. Finally the third case (C): $a \in A_2 \setminus A_1$ is fully symmetric to (B) .

(iii) Let $[(K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2)]_s \xrightarrow{a}$ for $a \in A$. According to the coinductive definition of the supervised product with partial observations repeatedly applied one of the three cases below occurs:

$$[(K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2)]_{sa} = \begin{cases} (1) & (K_1 \parallel K_2)_{sa}/^{A_o}_{A_u}(L_1 \parallel L_2)_{sa} \\ (2) & \cup_{\{s':\, P(s')=P(s)\}}(K_1 \parallel K_2)_{s'a}/^{A_o}_{A_u}(L_1 \parallel L_2)_{sa} \\ (3) & 0/^{A_o}_{A_u}(L_1 \parallel L_2)_{sa} \end{cases}$$

Now we treat the different cases separately. Case (1) is very easy. Consider the subcases $(1A) : a \in A_1 \cap A_2$ and $(1B) : a \in A_1 \setminus A_2$. The subcase $(1C) : a \in A_2 \setminus A_1$ is symmetric to the subcase (1B) and is therefore omitted. For (1A) we have $K_1 \xrightarrow{s_1a}$, $L_1 \xrightarrow{s_1a}$, $K_2 \xrightarrow{s_2a}$, $L_2 \xrightarrow{s_2a}$ with $P_1(s) = s_1$ and $P_2(s) = s_2$, i.e. $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \xrightarrow{s_1a}$ and $(K_2/^{A_{o,2}}_{A_{2u}} L_2) \xrightarrow{s_2a}$. Thus $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2) \xrightarrow{sa}$ and from the definition of R trivially $\langle[(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2)]_{sa}, [(K_1 \parallel K_2)/^{A_o}_{A_u}(L_1 \parallel L_2)]_{sa}\rangle \in R$.

For (1B) we have $K_1 \xrightarrow{s_1a}$, $L_1 \xrightarrow{s_1a}$, i.e. $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \xrightarrow{s_1a}$ and therefore
$[(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2)]_s \xrightarrow{a} [(K_1/^{A_{o,1}}_{A_{1u}} L_1)]_{s_1a} \parallel [(K_2/^{A_{o,2}}_{A_{2u}} L_2)]_{s_2}$.

In case (2) we again distinguish subcases (2A), (2B), (2C). For (2A) we have $\exists s' \in K^2 = (K_1 \parallel K_2)^2$ such that $P(s') = P(s)$, $K_1 \xrightarrow{s_1'a}$, $K_2 \xrightarrow{s_2'a}$, $L_1 \xrightarrow{s_1a}$, and $L_2 \xrightarrow{s_2a}$ with $P_1(s') = s_1'$ and $P_2(s') = s_2'$. Notice that $P(s) = P(s')$ implies by Lemma 5.10 that $P_1^{loc}P_1(s) = P_1^{loc}P_1(s')$ and $P_2^{loc}P_2(s) = P_2^{loc}P_2(s')$. But this means that $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \xrightarrow{s_1a}$ and $(K_2/^{A_{o,2}}_{A_{2u}} L_2) \xrightarrow{s_2a}$. Thus $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2) \xrightarrow{sa}$. The subcase (2B) is even easier. It is sufficient to show that $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \xrightarrow{s_1a}$, but this follows in the same way as in (2A). (2C) is symmetric to the subcase (2B).

For case (3): $a \in A_u$, we have again three obvious subcases. In subcase (3A): $a \in A_1 \cap A_2$ according to our assumption $A_{2u} \cap A_1 = A_2 \cap A_{1u}$ we have $a \in A_{1u} \cap A_{2u}$, which together with $L_1 \xrightarrow{s_1a}$ and $L_2 \xrightarrow{s_2a}$ gives $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \xrightarrow{s_1a}$ and $(K_2/^{A_{o,2}}_{A_{2u}} L_2) \xrightarrow{s_2a}$. Thus $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2) \xrightarrow{sa}$. In the subcase (3B) $(a \in A_1 \setminus A_2)$ $a \in A_u$ implies $a \in A_{1u}$, i.e. $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \xrightarrow{s_1a}$ and $(K_1/^{A_{o,1}}_{A_{1u}} L_1) \parallel (K_2/^{A_{o,2}}_{A_{2u}} L_2) \xrightarrow{sa}$. The same conclusion is drawn by symmetry in the subcase (3C).

□

Remark 5.15. *Condition $A_c \subseteq A_o$ in the last theorem means that at the global level we have always optimal solutions (supremal normal and controllable sublanguages) to the supervisory control and observations problem, while in general this is not the case at the local levels, where local observability and local normality might differ. Notice that for the first inclusion corresponding to (ii) in the proof of bisimilarity $A_{2u} \cap A_1 = A_2 \cap A_{1u}$ is not needed, while for the other inclusion it is needed and the condition $A_c \subseteq A_o$ is not needed.*

We have the following consequence:

Corollary 5.16. *If $A_c \subseteq A_o$, $A_{2u} \cap A_1 = A_2 \cap A_{1u}$, and $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$ then K_i controllable and observable with respect to L_i, $i = 1, 2$ implies that $(K_1 \parallel K_2)$ controllable and observable with respect to $(L_1 \parallel L_2)$.*

Proof. It is sufficient to notice that local supervised products are equal to K_i, $i = 1, 2$, i.e. by commutativity we obtain that the global supervised product is equal to $K = K_1 \parallel K_2$, which means that K is controllable and observable with respect to L. □

We have shown in the previous sections a sufficient condition, called mutual controllability, for preserving optimality in the full observation modular control (i.e. commutativity of the supremal controllable sublanguage and the synchronous product of partial languages). A natural question arises: with respect to which conditions this can be generalized to modular control with partial observations. Since there is no local optimality in general, we assume $A_{c,i} \subseteq A_{o,i}$ and the problem is when does the supremal normal and controllable sublanguage commute with the synchronous product of languages. Unfortunately, it is not possible to the best of our knowledge to define the supremal normal sublanguage by coinduction. The argument is similar as for the antipermissive control policy (see [20]). Nevertheless, using suitable automata representations (state partition automata [48]) there is the following algorithm for the computation of the supremal normal sublanguage. This algorithm will be used in the coinductive proof of our main theorem.

We use the representations of languages K and L by automata S_1 and S, where S_1 is a subautomaton of S such that $Aux(S_1)$ is an equivalence relation. We have proven in [17] that the condition of S_1 being state-partition automaton [47] is stronger, i.e. it guarantees that $Aux(S_1)$ is an equivalence relation. But it is known how to construct such representations [9] or [47]. Let s_0 denote the common initial state of S_1 and S. The transition structure of S_1 and S is denoted by $\to_1$ and $\to$, respectively. In the following algorithm we compute a supremal $(L, P)-$normal sublanguage of K.

Recall that $A = A_o \cup A_{uo}$ is a partition of A into observable events (A_o) and unobservable (A_{uo}) events with the natural projection $P : A^* \to A_o^*$ that erases unobservable events. In this paper we deal with supremal normal sublanguages. Let us recall from [23] the basic definition.

Definition 5.17. *(Normality.) Let $K, L \in \mathcal{L}$: $K \subseteq L$. K is said to be (L, P)-normal if $K^2 = L^2 \cap P^{-1}(P(K^2))$.*

Now normal relations are recalled from [18].

Definition 5.18. *(Normal relation.) Given two (partial) automata $S_1 = (S_1, \langle o_1, t_1 \rangle)$ and $S = (S, \langle o, t \rangle)$ as above with common initial state $s_0 \in S$, a binary relation $N(S_1, S)$ on $S_1 \times S$ is called a* normal relation *if for any $\langle s, t \rangle \in N(S_1, S)$ the following items hold:*

(i) $\forall a \in A \, : \, s \xrightarrow{a}_1 s_a \quad \Rightarrow \quad t \xrightarrow{a} t_a$ *and* $\langle s_a, t_a \rangle \in N(S_1, S)$

(ii) $\forall u \in A_{uo}: \ t \xrightarrow{u} t_u \quad \Rightarrow \quad s \xrightarrow{u}_1 s_u$.

(iii) $\forall a \in A_o : t \xrightarrow{a} t_a$ *and* $(\exists s' : \langle s, s' \rangle \in Aux(S_1) : s' \xrightarrow{a}_1 s'_a) \ \Rightarrow \ s \xrightarrow{a}_1 s_a$.

Remark 5.19. *Recall that (ii) can also be expressed using the set* $\lfloor s \rfloor_{Aux(S_1)}$. *The condition* $\exists s' : \langle s, s' \rangle \in Aux(S_1)$ *and* $s' \xrightarrow{a}_1 s'_a$ *can be replaced by the simpler one* $\lfloor s \rfloor_{Aux(S_1)} \xrightarrow{a}_1$.

For $s \in S_1$ and $t \in S$ we write $s \approx_{N(S_1,S)} t$ whenever there exists a normal relation $N(S_1, S)$ on $S_1 \times S$ such that $\langle s, t \rangle \in N(S_1, S)$. It was proven in [18] that

Theorem 5.20. *A (partial) language* K *is* $(L, P)-$*normal iff* $s_0 \approx_{N(S_1,S)} s_0$.

Now we are ready to formulate the algorithm for computation of supremal $(L, P)-$normal sublanguages.

Algorithm 1. *Let automata* S_1 *and* S *representing* K *and* L*, respectively, be such that* S_1 *is a subautomaton of* S *and* S_1 *is a state-partition automaton. Let us construct partial automaton* $\tilde{S} = \langle \tilde{o}, \tilde{t} \rangle$*, subautomaton of* S_1*, with* $\tilde{t}$ *denoted by* $\rightarrow'$.
*Define the auxiliary condition (*) consisting of (*a) and (*b) as follows:*
*(*a) if* $a \in A_{uo}$ *then* $\forall u \in A_{uo}^*$*:* $s_a \xrightarrow{u} \Rightarrow \ s_a \xrightarrow{u}_1$*;*
*(*b) if* $a \in A_o$ *then* $\forall s' \approx_{Aux(S_1)} s$ *:* $s' \xrightarrow{a} \ \Rightarrow \ s' \xrightarrow{a}_1$*, in which case also* $\forall u \in A_{uo}^*$*:* $s'_a \xrightarrow{u} \Rightarrow \ s'_a \xrightarrow{u}_1$.
Below are the steps of the algorithm.
1. Put $\tilde{S} := \{s_0\}$.
2. For any $s \in \tilde{S}$ *and* $a \in A$ *we put* $s \xrightarrow{a}' s_a$ *if* $s \xrightarrow{a}_1$ *and condition (*) is satisfied and we put in the case* $s \xrightarrow{a}'$ *also* $\tilde{S} := \tilde{S} \cup \{s_a\}$.
3. For any $s \in \tilde{S}$ *we put* $\tilde{o}(s) = o(s)$.

Let us denote by $\tilde{l} : \tilde{S} \to \mathcal{L}$ the unique (behavior) homomorphism given by finality of $\mathcal{L}$.

Theorem 5.21. $\tilde{l}(s_0)$ *is the supremal* $(L, P)-$*normal sublanguage of* K.

Proof. To prove the normality of $\tilde{l}(s_0)$ we show that the following relation is a normal relation on $\tilde{S} \times S$.

$$N = \{\langle (s_0)_u, (s_0)_u \rangle \mid u \in \tilde{l}(s_0)^2 \ \}.$$

Then $\tilde{l}(s_0)$ is normal with respect to L and P according to Theorem 5.20. Take a pair $\langle (s_0)_v, (s_0)_v \rangle \in N$ for some $v \in \tilde{l}(s_0)^2$.

(i) If $(s_0)_v \xrightarrow{a}'$ for $a \in A$, then clearly by construction of Algorithm 1 $(s_0)_v \xrightarrow{a}$. It is clear from the definition of N that $\langle (s_0)_{va}, (s_0)_{va} \rangle \in N$.

(ii) Let $a \in A_{uo}$ be such that $(s_0)_v \xrightarrow{a}$. We must show that $(s_0)_v \xrightarrow{a}'$, i.e. $\forall u \in A_{uo}^*$: $(s_0)_{va} \xrightarrow{u} \ \Rightarrow \ (s_0)_{va} \xrightarrow{u}_1$. It follows from $(s_0) \xrightarrow{v}'$ and Algorithm 1 that $\forall u \in A_{uo}^*$: $(s_0)_v \xrightarrow{u} \Rightarrow \ (s_0)_v \xrightarrow{u}_1$. Indeed, if we assume $v = v_1 \dots v_k$, for some $k \in Z$, then either $v_k \in A_{uo}$, i.e. $(s_0)_{v_1 \dots v_{k-1}} \xrightarrow{v_k}'$ means directly that $\forall u \in A_{uo}^*$: $(s_0)_v \xrightarrow{u} \Rightarrow \ (s_0)_v \xrightarrow{u}_1$ or $v_k \in A_o$, but then the condition (*) is even stronger: by putting $s' = s$ we obtain the same conclusion. Since in both cases $au \in A_{uo}^*$, the required implication holds as well for $(s_0)_{va}$ as required for $(s_0)_v \xrightarrow{a}'$.

(iii) Let $a \in A_o$ be such that $(s_0)_v \xrightarrow{a}$ and let there exist $s' \in \tilde{S}$: $s' \approx_{Aux(\tilde{S})} (s_0)_v$ with $s' \xrightarrow{a}'$. By Lemma 5.4 there exist two strings $w, w' \in A^*$ such that $P(w) = P(w')$, $(s_0)_v = (s_0)_w$, and $s' = (s_0)_{w'} \xrightarrow{a}$. According to the construction of Algorithm 1 for any $s \approx_{Aux(S_1)} (s_0)_{w'}$ there must be $s \xrightarrow{a} \Rightarrow \quad s \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo}^*$: $s_a \xrightarrow{u} \Rightarrow \quad s_a \xrightarrow{u}_1$. In order to show that $(s_0)_v \xrightarrow{a}{}'$ it must be that for any $q \approx_{Aux(S_1)} (s_0)_v$ there must be $q \xrightarrow{a} \Rightarrow \quad q \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo}^*$: $q_a \xrightarrow{u} \Rightarrow \quad q_a \xrightarrow{u}_1$. Now we use two facts. Firstly the fact that $Aux(S_1)$ is transitive (because S_1 is a state-partition automaton, a stronger condition), and secondly the fact that $s' \approx_{Aux(\tilde{S})} (s_0)_v$ implies that $s' \approx_{Aux(S_1)} (s_0)_v$. We obtain that $\langle s', q\rangle \in Aux(S_1)$. But this just means that for any $q \approx_{Aux(S_1)} (s_0)_v$ we have $q \xrightarrow{a} \Rightarrow \quad q \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo}^*$: $q_a \xrightarrow{u} \Rightarrow \quad q_a \xrightarrow{u}_1$, i.e. $(s_0)_v \xrightarrow{a}{}'$. Therefore N is a normality relation.

We show finally that the supremal $(L, P)-$normal sublanguage of K is contained in $\tilde{l}(s_0)$. Let N be a $(L, P)-$normal sublanguage of K. Then it is sufficient to show that

$$R = \{\langle N_u, \tilde{l}(s_0)_u\rangle \mid u \in N^2\}$$

satisfies (ii) of simulation relation in order to prove that $N^2 \subseteq \tilde{l}(s_0)^2$. Take an arbitrary pair $\langle N_w, \tilde{l}(s_0)_w\rangle \in R$ for some $w \in N^2$. Let $N_w \xrightarrow{a}$ for $a \in A$. Then also $K_w \xrightarrow{a}$, since $N \subseteq K$, and $L_w \xrightarrow{a}$ as well. This means that $(s_0)_w \xrightarrow{a}_1$ and $(s_0)_w \xrightarrow{a}$. In order to show that $\tilde{l}(s_0)_w \xrightarrow{a}$, i.e. $(s_0)_w \xrightarrow{a}{}'$, it must be shown that the condition (*) is satisfied.

For $a \in A_{uo}$ we need to show that $\forall u \in A_{uo}^*$: $(s_0)_{wa} \xrightarrow{u} \Rightarrow \quad (s_0)_{wa} \xrightarrow{u}_1$. But this is easy: $(s_0)_{wa} \xrightarrow{u}$ means $wau \in L^2$. Since N is $(L, P)-$normal, $wa \in N^2$ and $P(wa) = P(wau)$, we deduce $wau \in N^2 \subseteq K^2$. But this just means that $(s_0)_{wa} \xrightarrow{u}_1$.

For $a \in A_o$ it must be checked that for any $q \approx_{Aux(S_1)} (s_0)_w$: $q \xrightarrow{a} \quad \Rightarrow \quad q \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo}^*$: $q_a \xrightarrow{u} \Rightarrow \quad q_a \xrightarrow{u}_1$. There exist v, v' : $P(v) = P(v')$ such that $q = (s_o)_{v'}$ and $(s_0)_w = (s_0)_v$. Since S_1 is a state-partition automaton and $(s_0)_w = (s_0)_v$ is in two states of the observer automaton, we conclude by the property of state-partition automaton that these two states of the observer automaton coincide. But this means that there exists $w' \in A^*$ such that $P(w) = P(w')$ and $q = (s_0)_{w'}$. Now $q \xrightarrow{a}$ means that $w'a \in L^2$. By normality of N it follows from $wa \in N^2$ and $w'a \in L^2$ that $w'a \in N^2$. Therefore $w'a \in K^2$ (because $N \subseteq K$), which means that $q \xrightarrow{a}_1$. The rest is similar as for $a \in A_{uo}$: if for $u \in A_{uo}^*$: $q_a = (s_o)_{w'a} \xrightarrow{u}$, then $w'au \in L^2$, by normality of N and using $wa \in N^2$, where $P(w'au) = P(wa)$ we have $w'au \in N^2 \subseteq K^2$. But this just means that $(s_0)_{w'a} = q_a \xrightarrow{u}_1$.

We conclude that $\tilde{l}(s_0)_w \xrightarrow{a}$ and R satisfies (ii) of simulation relation, i.e. we have the inclusion $N^2 \subseteq \tilde{l}(s_0)^2$. Note that since N was arbitrary $(L, P)-$normal sublanguage of K, and $\tilde{l}(s_0)$ has been shown to be a $(L, P)-$ normal sublanguage of K, it follows that $\tilde{l}(s_0)$ is the supremal $(L, P)-$normal sublanguage of K. ☐

In our main theorem a condition similar to mutual controllability (see [44]) is needed. We call it by analogy also mutual normality.

Definition 5.22. *Given partial languages $L_i = (L_i^1, L_i^2)$ and $L_j = (L_j^1, L_j^2)$, L_i and L_j are said to be mutually normal if*

$$(P_i^{loc})^{-1} P_i^{loc}(L_i^2) \cap P_i(P_j)^{-1}(L_j^2) \subseteq L_i^2 \text{ and}$$
$$(P_j^{loc})^{-1} P_j^{loc}(L_j^2) \cap P_j(P_i)^{-1}(L_i^2) \subseteq L_j^2.$$

Mutual normality can be viewed as normality of the local plant languages with respect to the local views of the other plant languages. In order to ensure the full compatibility of the global and local observations, a property called decomposability, similar but slightly stronger that the one known from the decentralized control ([36]), is needed. Section 5 contains an example, where mutual normality does not hold and a procedure for verification of this property.

The following lemma provides an insight into the relationship between local and global observations in our setting.

Lemma 5.23. *If $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$ then the following diagram commutes, i.e. $\forall s \in A*$:*

$$P_1P(s) = P_1^{loc}P_1(s) \text{ and } P_2P(s) = P_2^{loc}P_2(s).$$

$$\begin{tikzcd}
 & A^* \arrow[dl, "P_1"'] \arrow[dd, "P"'] \arrow[dr, "P_2"] & \\
A_1^* \arrow[dd, "P_1^{loc}"'] & & A_2^* \arrow[dd, "P_2^{loc}"'] \\
 & A_o^* \arrow[dl, "P_1"'] \arrow[dr, "P_2"] & \\
A_{o,1}^* & & A_{o,2}^*
\end{tikzcd}$$

Proof. We will show by induction that $P_1^{loc}P_1 = P_1P$, i.e. $\forall s \in A^* : \ P_1^{loc}P_1(s) = P_1P(s)$. For $s = \varepsilon$ it is easy: $P_1^{loc}P_1(\varepsilon) = \varepsilon = P_1P(\varepsilon)$. Assume now that the equality holds for $s \in A^*$. We must show that it holds also for sa with $a \in A$ arbitrary. Since all projections involved are catenative, it is sufficient to prove the lemma for $s = a \in A$ arbitrary. For $s = a$ several cases must be distinguished. If $a \in A_1 \cap A_2$ then P_1 and P_2 are identity on a and according to the assumption on the structure of (locally) observable event sets $P_1^{loc}(a) = P(a) = P_2^{loc}(a)$, whence $P_1^{loc}P_1(a) = P_1P(a)$. For $a \in A_1 \setminus A_2$ we have $P_1^{loc}(a) = P(a)$, whence $P_1^{loc}P_1(a) = P_1P(a)$. Finally for $a \in A_2 \setminus A_1$ clearly $P_1^{loc}P_1(a) = P_1P(a) = \varepsilon$. In the same way it can be shown that $P_2^{loc}P_2 = P_2P$. ☐

The property of decomposability has been introduced in Definition 5.1. For instance, $P(L) \subseteq A_o^*$ is decomposable with respect to P_1 and P_2 if $P(L) = P_1^{-1}P_1P(L) \cap P_2^{-1}P_2P(L)$. Using the property of decomposability applied to projected language we obtain:

Lemma 5.24. *Let $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$, $L = L_1 \parallel L_2$, and let $PL = P(L_1 \parallel L_2)$ be decomposable with respect to P_1 and P_2, $s, s' \in L$. Then $P(s) = P(s')$ iff $P_1^{loc}P_1(s) = P_1^{loc}P_1(s')$ and $P_2^{loc}P_2(s) = P_2^{loc}P_2(s')$.*

Proof. Under the assumption that $PL = P(L_1 \parallel L_2)$ is decomposable with respect to P_1 and P_2 the statement is an easy consequence of Lemma 5.23. Indeed, $P(L) = P_1^{-1}P_1P(L) \cap P_2^{-1}P_2P(L) = P_1^{-1}P_1^{loc}P_1(L) \cap P_2^{-1}P_2^{loc}P_2(L)$. Thus, for $s \in L$:
$P(s) = P_1^{-1}P_1^{loc}P_1(s) \cap P_2^{-1}P_2^{loc}P_2(s)$. It is now easy to see that ($P_1^{loc}P_1(s) = P_1^{loc}P_1(s')$ and $P_2^{loc}P_2(s) = P_2^{loc}P_2(s')$) iff $P(s) = P(s')$. Since $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$, the backward implication follows from Lemma 5.10. The forward implication follows from the formula above. Thus $P_1^{loc}P_1(s) = P_1^{loc}P_1(s')$ and $P_2^{loc}P_2(s) = P_2^{loc}P_2(s')$ iff $P(s) = P(s')$. ☐

The last Lemma can be viewed as a weak version of the converse implication to Lemma 5.10. Note however that Lemma 5.24 is not needed in our main theorem. The following lemma will be used in the proof of the main theorem.

Lemma 5.25. *Assume that $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$. Let $v \in A^*$ with $P_1(v) = v_1 \in A_1^*$ and $P_2(v) = v_2 \in A_2^*$. Let $v_1' \in A_1^*$ be such that $P_1^{loc}(v_1) = P_1^{loc}(v_1')$. Then there exists $v' \in A^*$ such that $P_1(v') = v_1'$, $P_2^{loc}P_2(v') = P_2^{loc}(v_2)$, and $P(v) = P(v')$.*

Proof. It can be proven by structural induction with respect to $v \in A^*$. For $v = \varepsilon$ we have $v_1 = P_1(v) = \varepsilon$. Therefore there must be $v_1' \in A_{uo,1}^*$. Without loss of generality we take an arbitrary, but fixed $v_1' = u_1 \dots u_k$ with $u_1, \dots, u_k \in A_{uo,1}$. Since we require $P(v) = P(v')$, the choice for v is $v \in A_{uo}^*$. Furthermore, we require $v' \in P_1^{-1}(v_1')$, hence v' is of the form: $(A_2 \setminus A_1)^* u_1 (A_2 \setminus A_1)^* \dots (A_2 \setminus A_1)^* u_k (A_2 \setminus A_1)^* \ \cap \ A_{uo}^* =$
$(A_{2,uo} \setminus A_1)^* u_1 (A_{2,uo} \setminus A_1)^* \dots (A_{2,uo} \setminus A_1)^* u_k (A_{2,uo} \setminus A_1)^*$.
Finally we need to ensure that $P_2^{loc} P_2(v') = P_2^{loc}(v_2)$. Since $v_2 = P_2(v) = \varepsilon$, it amounts to ensure that $P_2(v') \in A_{uo,2}^*$. Clearly $A_{uo,2} \setminus A_1 \subseteq A_{uo,2}^*$, but we need also $P_2(u_i) \in A_{uo,2}$ for all $i \in \{1, \dots, k\}$. This is satisfied due to our assumption that shared events have the same observation status for both modules: if $\exists i : \ u_i \in A_1 \cap A_2$, then $u_i \in A_{uo,2}$, because $u_i \in A_{uo,1}$. Therefore we have the following set for the choice of v' :
$(A_{2,uo} \setminus A_1)^* u_1 (A_{2,uo} \setminus A_1)^* \dots (A_{2,uo} \setminus A_1)^* u_k (A_{2,uo} \setminus A_1)^*$.

The induction step follows: we assume that the lemma holds for $v \in A^*$ and we will show that it holds also for $v'a$. More precisely we suppose that for $v \in A^*$ and any $v_1' \in A_1^*$ such that $P_1^{loc}(v_1) = P_1^{loc}(v_1')$ there exists $v' \in A^*$ such that $P_1(v') = v_1'$ and $P_2^{loc} P_2(v') = P_2^{loc}(v_2)$. It is sufficient to take for $va \in A^*$ and v_1'' simply $v'' := v'a$ with v' corresponding to v according to the induction hypothesis, because all projections involved are catenative. This completes the proof of the lemma. $\square$

Let us introduce the notation $\sup N(K, L, P)$ for the supremal $(L, P)-$ normal sublanguage of K. Our main theorem follows.

Theorem 5.26. *(Sufficiency for modular equals global synthesis for supremal normal sublanguages) If $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$, and L_1 and L_2 are mutually normal, then*

$$\sup N(K_1, L_1, P_1^{loc}) \parallel \sup N(K_2, L_2, P_2^{loc}) = \sup N(K_1 \parallel K_2, L_1 \parallel L_2, P).$$

Proof. The coinductive proof principle will be used. We will 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. 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 $i \in \{1, 2\}$, 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. 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 $\tilde{l}_i : \tilde{S} \rightarrow \mathcal{L}$. It will be clear from the context whether local or global automaton is meant, i.e. this simplification of notation should not lead to any confusion.
The (common) initial state of S and T is denoted by s_0 and for $i = 1, 2$ 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, $\tilde{l}(s_0) = supN(K, L, P)$ and $\tilde{l}_i(s_0^i) = supN(K_i, L_i, P_i^{loc})$, where $K = K_1 \parallel K_2$ and $L = L_1 \parallel L_2$.

We show that

$$R = \{\langle [\tilde{l}(s_0)]_v, [\tilde{l}_1(s_0^1) \parallel \tilde{l}_2(s_0^2)]_v \rangle \mid v \in (\tilde{l}(s_0))^2\}$$

is a bisimulation relation, from which the claim of the theorem follows by coinduction. Take a $v \in (\tilde{l}(s_0))^2$ arbitrary, but fixed.

(i) is trivial, Algorithm 1 does not consider marking components.

(ii) Let $[\tilde{l}(s_0)]_v \xrightarrow{a}$, i.e. condition (*) of Algorithm 1 is satisfied. It must be shown that $[\tilde{l}_1(s_0^1) \parallel \tilde{l}_2(s_0^2)]_v \xrightarrow{a}$.

First we assume that $a \in A_1 \cap A_2$. Then $[\tilde{l}_1(s_0^1) \parallel \tilde{l}_2(s_0^2)]_v = [\tilde{l}_1(s_0^1)]_{v_1} \parallel [\tilde{l}_2(s_0^2)]_{v_2}$ with $P_1(v) = v_1$ and $P_2(v) = v_2$. We show that $[\tilde{l}_1(s_0^1)]_{v_1} \xrightarrow{a}$, i.e. $(s_0^1)_{v_1} \xrightarrow{a}\!\!{}'$. According to Algorithm 1 applied to S_1 and T_1 we must show that condition (*) holds. First of all note that $(s_0^1)_{v_1} \xrightarrow{a}_1$. Indeed, $[\tilde{l}(s_0)]_v \xrightarrow{a}$ implies that $(s_0)_v \xrightarrow{a}_1$, i.e. $va \in K^2 = (K_1 \parallel K_2)^2$. Therefore $v_1 a = P_1(va) \in K_1^2 \subseteq L_1^2$, i.e. $(s_0^1)_{v_1} \xrightarrow{a}_1$. Similarly $v_2 a = P_2(va) \in K_2^2 \subseteq L_2^2$.

If $a \in A_{uo,1} \subseteq A_{uo}$ then it must be shown that $\forall u_1 \in A_{uo,1}^*$: $(s_0^1)_{v_1 a} \xrightarrow{u_1} \Rightarrow (s_0^1)_{v_1 a} \xrightarrow{u_1}_1$. Let $(s_0^1)_{v_1 a} \xrightarrow{u_1}$, i.e. $v_1 a u_1 \in L_1^2$. We know that condition (*) holds for $\tilde{S}$, i.e. $\forall u \in A_{uo}^*$: $(s_0)_{va} \xrightarrow{u} \Rightarrow (s_0)_{va} \xrightarrow{u}_1$. We have $u_1 \in A_{uo,1}^* \subseteq A_{uo}^*$. It is sufficient to consider $u = u_1$ and notice that $vau \in P_1^{-1}(v_1 a u_1)$ (recall that $v_1 = P_1(v)$). We show that $vau \in L^2 = (L_1 \parallel L_2)^2$. Since $vau \in P_1^{-1}(v_1 a u_1)$, it follows that $vau \in P_1^{-1}(L_1^2)$. It remains to show that $vau \in P_2^{-1}(L_2^2)$. Mutual normality is used: $P_2(vau) = P_2(vau_1) = v_2 a P_2(u_1) \in (P_2^{loc})^{-1} P_2^{loc}(L_2^2) \cap P_2(P_1)^{-1}(L_1^2) \subseteq L_2^2$, where $v_2 a P_2(u_1) \in (P_2^{loc})^{-1} P_2^{loc}(L_2^2)$, because $v_2 a \in K_2^2 \subseteq L_2^2$ and $P_2(u_1) \in A_{uo,2}^*$ due to our assumption that $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$, i.e. $P_2^{loc}(v_2 a P_2(u_1)) = P_2^{loc}(v_2 a)$. Note that it can be that $P_2(u_1) = \varepsilon$ in the case $u_1 \in (A_1 \setminus A_2)^*$. Now, $vau \in L^2$, i.e. $(s_0)_{va} \xrightarrow{u}$, and from condition (*) holds for $\tilde{S}$ we obtain $(s_0)_{va} \xrightarrow{u}_1$. But this means that $vau = vau_1 \in K^2$, i.e. $v_1 a u_1 = P_1(vau) \in K_1^2$, because $P_1(u_1) = u_1 \in A_{uo,1}^*$. Hence $(s_0^1)_{v_1 a} \xrightarrow{u_1}_1$ as required by condition (*) of Algorithm 1.

If $a \in A_{o,1} \subseteq A_o$ then we know that $\forall s' \approx_{Aux(S)} (s_0)_v : s' \xrightarrow{a} \Rightarrow s' \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo}^*$: $s'_a \xrightarrow{u} \Rightarrow s'_a \xrightarrow{u}_1$. It must be shown that $(s_0^1)_{v_1} \xrightarrow{a}\!\!{}'$, i.e. $\forall q^1 \approx_{Aux(S_1)} (s_0^1)_{v_1} : q^1 \xrightarrow{a} \Rightarrow q^1 \xrightarrow{a}_1$, in which case also $\forall u_1 \in A_{uo,1}^*$: $q_a^1 \xrightarrow{u_1} \Rightarrow q_a^1 \xrightarrow{u_1}_1$. Let $q^1 \approx_{Aux(S_1)} (s_0^1)_v : q^1 \xrightarrow{a}$. Since S_1 is a state-partition automaton, there exists $v_1' \in A_1^*$ such that $P_1^{loc}(v_1') = P_1^{loc}(v_1)$ and $q^1 = (s_0^1)_{v_1'}$. For this $v_1' \in A_1^*$ and $v \in A^*$ above there exists according to Lemma 5.25 a $v' \in A^*$ such that $P_1(v') = v_1'$ satisfying moreover $P_2^{loc} P_2(v') = P_2^{loc}(v_2)$ and $P(v) = P(v')$. Since $q^1 \xrightarrow{a}_1$ we have $v_1' a \in K_1^2 \subseteq L_1^2$. Then $v'a \in P_1^{-1}(v_1' a)$. Therefore $v'a \in P_1^{-1}(L_1^2)$. We show that $v'a \in P_2^{-1}(L_2^2)$ using mutual normality. Indeed, $P_2(v'a) = v_2' a \in (P_2^{loc})^{-1} P_2^{loc}(L_2^2) \cap P_2(P_1)^{-1}(L_1^2) \subseteq L_2^2$, where $v_2' a \in (P_2^{loc})^{-1} P_2^{loc}(L_2^2)$, because $v_2 a \in K_2^2 \subseteq L_2^2$ and $P_2^{loc}(v_2' a) = P_2^{loc}(v_2 a)$ by Lemma 5.25. Therefore $v_2' a \in L_2^2$ by applying mutual normality. Also $v_2' a \in P_2(P_1)^{-1}(L_1^2)$, because $v_1' a \in L_1^2$. Thus, $v'a \in P_1^{-1}(L_1^2) \cap P_2^{-1}(L_2^2) = L^2$. Since $P(v') = P(v)$, we have $(s_0)_{v'} \approx_{Aux(S)} (s_0)_v$. 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 $(s_0^1)_{v_1'} = q^1 \xrightarrow{a}_1$, because $v'a \in K^2 = (K_1 \parallel K_2)^2$ implies that $v_1' a = P_1(v'a) \in K_1^2$. We show also that $\forall u_1 \in A_{uo,1}^*$: $q_a^1 \xrightarrow{u_1} \Rightarrow q_a^1 \xrightarrow{u_1}_1$. Indeed, $q_a^1 \xrightarrow{u_1}$ means $v_1' a u_1 \in L_1^2$. Similarly as for $a \in A_{uo}$, by considering $u = u_1 \in A_{uo}^*$ we obtain using mutual normality that $v'au \in L^2$, i.e. $(s_0)_{v'a} \xrightarrow{u}$ whence $(s_0)_{v'a} \xrightarrow{u}_1$. But this means that $v'au \in K^2$, i.e. $v_1' a u_1 = P_1(v'au) \in K_1^2$ and $q_a^1 \xrightarrow{u_1}_1$.

In a symmetric way $[\tilde{l}_2(s_0^2)]_{v_2} \xrightarrow{a}$, i.e. $[\tilde{l}_1(s_0^1) \parallel \tilde{l}_2(s_0^2)]_v \xrightarrow{a}$. The cases $a \in A_1 \setminus A_2$ and $a \in A_2 \setminus A_1$ are simpler. We need to show that $[\tilde{l}_1(s_0^1)]_{v_1} \xrightarrow{a}$. The proof for this case follows the same lines as above, but it is much simpler due to $P_2(a) = \varepsilon$, i.e. in order to show e.g $vau \in P_2^{-1}(L_2^2)$ for $a \in A_{uo}$ it is sufficient to show $P_2(vu) = v_2 P_2(u) \in L_2^2$. Also the case $a \in A_o$ is simpler than for $a \in A_1 \cap A_2$.

(iii) Let $[\tilde{l}_1(s_0^1) \parallel \tilde{l}_2(s_0^2)]_v \xrightarrow{a}$. It must be shown that $[\tilde{l}(s_0)]_v \xrightarrow{a}$, i.e. condition (*) of Algorithm 1 is satisfied. According to the coinductive definition of synchronized product inductively applied we have: $[\tilde{l}_1(s_0^1) \parallel \tilde{l}_2(s_0^2)]_v = \tilde{l}_1(s_0^1)_{v_1} \parallel \tilde{l}_2(s_0^2)_{v_2}$ with $P_1(v) = v_1$ and $P_2(v) = v_2$. It follows that $\tilde{l}_1(s_0^1)_{v_1} \xrightarrow{a}$ and $\tilde{l}_2(s_0^2)_{v_2} \xrightarrow{a}$, i.e. $(s_0^1)_{v_1} \xrightarrow{a}{}'$ and $(s_0^2)_{v_2} \xrightarrow{a}{}'$. It must be shown that $[\tilde{l}(s_0)]_v \xrightarrow{a}$, which is equivalent to $(s_0)_v \xrightarrow{a}{}'$, i.e. condition (*) of Algorithm 1 applied to (global) automata S and T is satisfied.

We know that
if $a \in A_{uo,1}$ then $\forall u \in A_{uo,1}^*$: $(s_0^1)_{v_1a} \xrightarrow{u} \Rightarrow (s_0^1)_{v_1a} \xrightarrow{u}_1$;
if $a \in A_{o,1}$ then $\forall s' \approx_{Aux(S_1)} (s_0^1)_{v_1}$: $s' \xrightarrow{a} \Rightarrow s' \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo,1}^*$: $s'_a \xrightarrow{u} \Rightarrow s'_a \xrightarrow{u}_1$.

We know also that
if $a \in A_{uo,2}$ then $\forall u \in A_{uo,2}^*$: $(s_0^2)_{v_2a} \xrightarrow{u} \Rightarrow (s_0^2)_{v_2a} \xrightarrow{u}_1$;
if $a \in A_{o,2}$ then $\forall s' \approx_{Aux(S_2)} (s_0^2)_{v_2}$: $s' \xrightarrow{a} \Rightarrow s' \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo,2}^*$: $s'_a \xrightarrow{u} \Rightarrow s'_a \xrightarrow{u}_1$.

We need to show that
if $a \in A_{uo}$ then $\forall u \in A_{uo}^*$: $(s_0)_{va} \xrightarrow{u} \Rightarrow (s_0)_{va} \xrightarrow{u}_1$;
if $a \in A_o$ then $\forall s' \approx_{Aux(S)} (s_0)_v$: $s' \xrightarrow{a} \Rightarrow s' \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo}^*$: $s'_a \xrightarrow{u} \Rightarrow s'_a \xrightarrow{u}_1$.

First we assume that $a \in A_1 \cap A_2$ and $a \in A_{uo}$. Let $u \in A_{uo}^*$: $(s_0)_{va} \xrightarrow{u}$. This means that $vau \in L^2$. Therefore if $v_i := P_i(v)$, then $v_i a P_i(u) \in L_i^2$, $i = 1, 2$. But this means that $(s_0^i)_{v_ia} \xrightarrow{P_i(u)}$ for $i = 1, 2$. Since $P_1(u) \in A_{uo,1}^*$ we obtain $(s_0^1)_{v_1a} \xrightarrow{P_1(u)}_1$. Similarly we obtain $(s_0^2)_{v_2a} \xrightarrow{P_2(u)}_1$. Thus, $P_i(vau) \in K_i^2$ for $i = 1, 2$, i.e. $vau \in P_1^{-1}(K_1^2) \cap P_2^{-1}(K_2^2) = K^2$, which means that $(s_0)_{va} \xrightarrow{u}_1$.

Let $a \in A_o$ and $s' \approx_{Aux(S)} (s_0)_v$: $s' \xrightarrow{a}$. Since S is a state-partition automaton, there exists $v' \in K^2 \subseteq L^2$: $P(v') = P(v)$ and $s' = (s_0)_{v'}$. Thus $s' \xrightarrow{a}$ is equivalent to $v'a \in L^2$. Therefore $P_i(v'a) := v'_i a \in L_i^2$. Using Lemma 5.10 we have $P_i^{loc}(v'_i) = P_i^{loc}(v_i)$, $i = 1, 2$, i.e. $s'_i := (s_0^i)_{v'_i} \approx_{Aux(S_i)} (s_0^i)_{v_i}$, where $s'_i \xrightarrow{a}$, $i = 1, 2$. Hence according to our assumption $s'_i \xrightarrow{a}_1$ $i = 1, 2$, and moreover $\forall u_i \in A_{uo,i}^*$: $(s'_i)_a \xrightarrow{u} \Rightarrow (s'_i)_a \xrightarrow{u}_1$. This means that $P_i(v'a) = v'_i a \in K_i^2$, $i = 1, 2$, hence $v'a \in P_1^{-1}(K_1^2) \cap P_2^{-1}(K_2^2) = K^2$, which is equivalent to $s' \xrightarrow{a}_1$. Moreover, in this case we obtain for $u \in A_{uo}^*$: $s'_a \xrightarrow{u}$ and for $i = 1, 2$: $v'au \in L^2$, $v'_i a P_i(u) \in L_i^2$, i.e. $(s'_i)_a \xrightarrow{P_i(u)}$, which implies according to our assumption that $(s'_i)_a \xrightarrow{P_i(u)}_1$ for $i = 1, 2$. But this means that $v'_i a P_i(u) \in K_i^2$ for $i = 1, 2$, i.e. $v'au \in P_1^{-1}(K_1^2) \cap P_2^{-1}(K_2^2) = K^2$, which is equivalent to $s'_a \xrightarrow{u}_1$. This proves that $[\tilde{l}(s_0)]_v \xrightarrow{a}$ for $a \in A_1 \cap A_2$.

If $a \in A_1 \setminus A_2$, then we only have $\tilde{l}_1(s_0^1)_{v_1} \xrightarrow{a}$, i.e. the condition (*) of Algorithm 1 is satisfied for S_1 subautomaton of T_1:
if $a \in A_{uo,1}$ then $\forall u \in A_{uo,1}^*$: $(s_0^1)_{v_1a} \xrightarrow{u} \Rightarrow (s_0^1)_{v_1a} \xrightarrow{u}_1$;
if $a \in A_{o,1}$ then $\forall s' \approx_{Aux(S_1)} (s_0^1)_{v_1}$: $s' \xrightarrow{a} \Rightarrow s' \xrightarrow{a}_1$, in which case also $\forall u \in A_{uo,1}^*$: $s'_a \xrightarrow{u} \Rightarrow s'_a \xrightarrow{u}_1$. Nevertheless, this is still sufficient to prove that the condition (*) of Algorithm 1 is satisfied for S subautomaton of T, because $a \notin A_2$. The proof is in this case very similar. For instance, if $a \in A_{uo}$ then we must show that $\forall u \in A_{uo}^*$: $(s_0)_{va} \xrightarrow{u} \Rightarrow (s_0)_{va} \xrightarrow{u}_1$. Let $u \in A_{uo}^*$: $(s_0)_{va} \xrightarrow{u}$. This means that $vau \in L^2$, i.e. $v_1 a P_1(u) = P_1(vau) \in L_1^2$ and $v_2 P_2(u) = P_2(vau) \in L_2^2$. We obtain $(s_0^1)_{v_1a} \xrightarrow{P_1(u)}$. It follows that $(s_0^1)_{v_1a} \xrightarrow{P_1(u)}_1$, which amounts to $v_1 a P_1(u) = P_1(vau) \in K_1^2$. It remains to show that $P_2(vau) = v_2 P_2(u) \in K_2^2$. It follows from $[\tilde{l}_2(s_0^2)] \xrightarrow{v_2}$ and from $v_2 P_2(u) \in L_2^2$ that $v_2 P_2(u) = P_2(vau) \in K_2^2$, i.e. $(s_0)_{va} \xrightarrow{u}_1$. Similar arguments are used for verification of condition (*) from Algorithm 1 in the case $a \in A_o$.

□

Remark 5.27. *Notice also that for (iii) in the above proof no assumption is used (except $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$ that is needed for Lemma 5.10). This means that under very general conditions we have one inclusion:*

Corollary 5.28. *If $A_{o,2} \cap A_1 = A_2 \cap A_{o,1}$, then we have*

$$\sup N(K_1, L_1, P_1^{loc}) \parallel \sup N(K_2, L_2, P_2^{loc}) \subseteq \sup N(K_1 \parallel K_2, L_1 \parallel L_2, P).$$

Note that Theorem 5.26 is useful for the computation of (global) supremal normal sublanguages of large distributed plants. If the conditions of the theorem are satisfied, then it is sufficient to compute local supremal normal sublanguages and synchronize them.

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

Note that an extension of our results from $n = 2$ to an arbitrary number $n \in N$ of local modules is quite straightforward and thus omitted in this paper. The condition of mutual normality between any pair of local plants is required. The corresponding theorem is:

Theorem 5.29. *If for any pairs $i, j \in \{1, \ldots, n\}$: $i \neq j$, $A_{o,j} \cap A_i = A_j \cap A_{o,i}$, and L_i and L_j are mutually normal, then $\|_{i=1}^n \sup N(K_i, L_i, P_i^{loc}) = \sup N(\|_{i=1}^n K_i, \|_{i=1}^n L_i, P)$.*

In [44] there is a procedure to change a plant which does not satisfy the mutual controllability condition into another 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, in this case mutual normality is not needed.

Theorem 5.30. Shuffle case in 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*

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

Proof. Note that in the shuffle case mutual controllability is trivially satisfied and 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, $\tilde{l}(s_0) = supN(K, L, P)$ and for any $i \in \mathbb{Z}_n$: $\tilde{l}_i(s_0^i) = supN(K_i, L_i, P_i^{loc})$. 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. Recall that only one simulation (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 then it must be shown 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 $\forall s' \approx_{Aux(S)} (s_0)_v$: $s' \xrightarrow{a} \Rightarrow s' \xrightarrow{a}_1$, 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. $\forall q^i \approx_{Aux(S_i)} (s_0^i)_{v_i}$: $q^i \xrightarrow{a}_i \Rightarrow\ q^i \xrightarrow{a}_{1i}$, 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 $q^i \approx_{Aux(S_i)} (s_0^i)_v$: $q^i \xrightarrow{a}_i$. Since S_i is a state-partition automaton, there exists $v'_i \in A_i^*$ such that $P_i^{loc}(v'_i) = P_i^{loc}(v_i)$ 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 $(s_0)_{v'} \approx_{Aux(S)} (s_0)_v$. 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 $v'_i a u_1 \in L_i$. 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. $v'_i a u_i = P_i(v'au) \in K_i$, or equivalently $q_a^i \xrightarrow{u_i}_{1i}$. $\square$

In supervisory control of partially observed DES one is interested in computation of supremal controllable and normal sublanguages. The question is whether the results of Theorems 4.5 and 5.26 can be combined. We have shown it in [21] using a single step algorithm for computation of supremal controllable and normal sublanguages and the following result holds. The notation $\sup \mathrm{CN}(K, L, P, A_u)$ is chosen for the supremal controllable with respect to L and A_u and $(L, P)-$normal sublanguage of K.

Theorem 5.31. Modular control synthesis equals global control synthesis for supremal controllable and normal sublanguage in 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

$$\|_{i=1}^n \sup \mathrm{CN}(K_i, L_i, P_i^{loc}, A_{iu}) = \sup \mathrm{CN}(\|_{i=1}^n K_i, \|_{i=1}^n L_i, P, A_u). \tag{2}$$

6 Examples and Verification of Mutual Normality

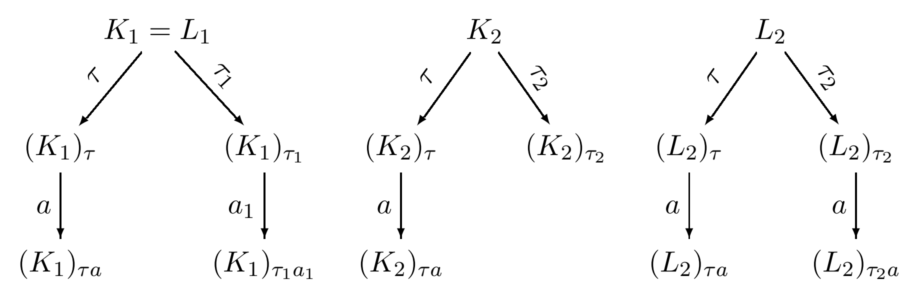
The purpose of this section is mainly to illustrate our results with examples. Before starting with concrete examples we consider several extreme cases of distributed DES. First of all, if all event alphabets are disjoint, the so called shuffle case, we notice that $P_i(P_j)^{-1}(L_j^2) = A_i^*$ for any $L_j^2 \subseteq A_j^*$. This means that the condition of mutual normality cannot be satisfied. The intuitive reason is that there is no interconnection between local subsystems in this case. This is not surprising, because the observations of local agents are in this case completely independent and therefore there is a huge gap between local and global observations.

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 trivialy satified. 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.

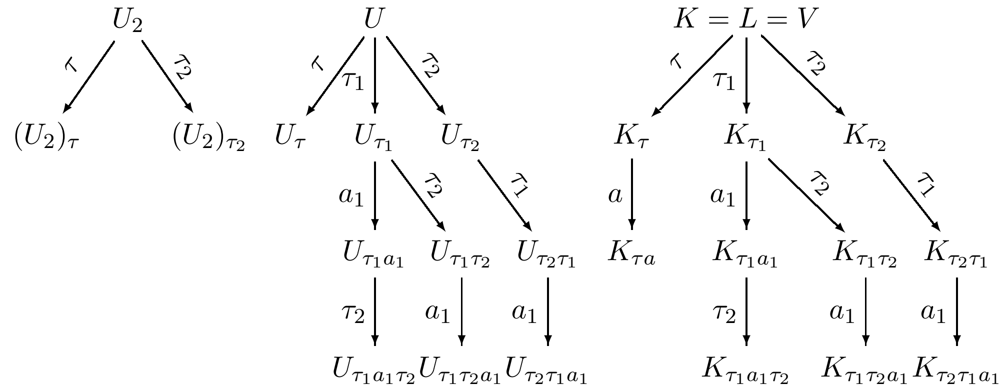
6.1 Examples

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

Example 6.1. *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 local specification and plant languages, where only second (prefix-closed) components are considered:*



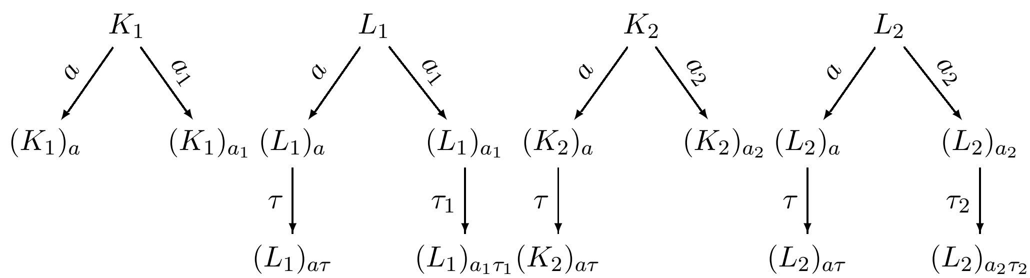
We use the notation $U_1 = \sup N(K_1, L_1, P_1^{loc})$, $U_2 = \sup N(K_2, L_2, P_2^{loc})$, $U = \sup N(K_1, L_1, P_1^{loc}) \parallel \sup N(K_2, L_2, P_2^{loc})$, and $V = \sup N(K_1 \parallel K_2, L_1 \parallel L_2, P)$. We have trivially that $U_1 = K_1 = L_1$. It is easy to see that $U_2^2 = \sup N(K_2, L_2, P_2^{loc})^2 = \{\varepsilon, \tau, \tau_2\}$. 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:*



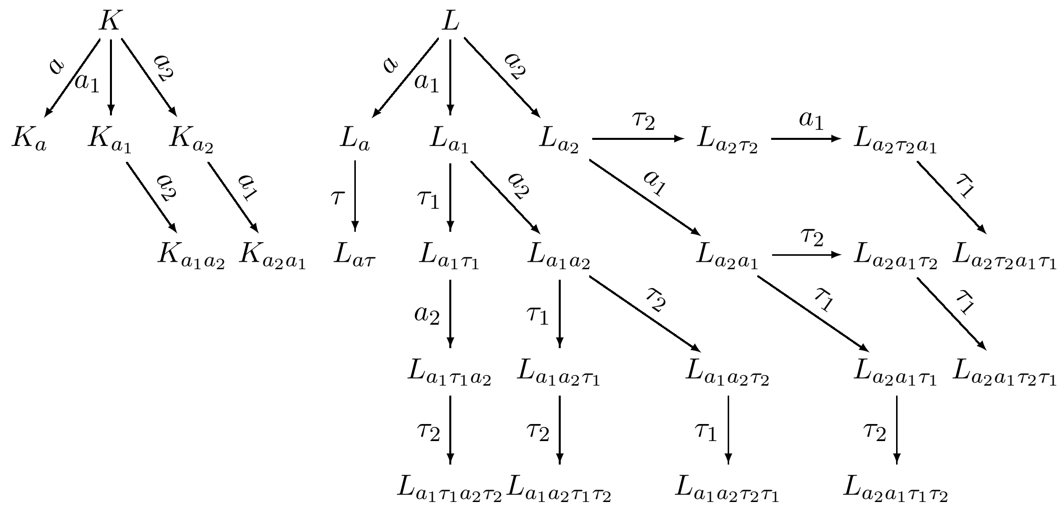
Thus, $U \neq V$, because $U_\tau \overset{a}{\not\to}$, while $V_\tau \overset{a}{\to}$. Therefore we only have the strict inclusion $U \subset V$ and the commutativity studied in this paper does not hold for this example. According to theorem 5.26 the mutual normality cannot hold. Indeed, we have $(P_1^{loc})^{-1}P_1^{loc}(L_1^2) \cap P_1(P_2)^{-1}(L_2^2) = \tau_1^*(\tau\tau_1^* a_1 + a_1\tau_1^*\tau)\tau_1^*$, but we have e.g. $(\tau_1)^n \notin L_1^2$ for $n \geq 2$!

Next an example is given, where the commutativity between the supremal normal sublanguages and the parallel product holds without the mutual normality condition. Therefore mutual normality is not a necessary condition for the commutativity. This should not be surprising, because mutual normality as a structural condition concerns only local open-loop languages and not local specification languages.

Example 6.2. *Consider the same event alphabets as in the preceding example and the following local specification and plant languages:*



Computing parallel products $K = K_1 \parallel K_2$ and $L = L_1 \parallel L_2$ yields:



In this example we have

$$\sup N(K_1, L_1, P_1^{loc})^2 = \{\varepsilon\} = \sup N(K, L, P)^2,$$

and $\sup N(K_2, L_2, P_2^{loc})^2 = \{\varepsilon, a, a\tau\}$, *i.e. the commutativity holds true. On the other hand, mutual normality does not hold for the same reason as above:*

$$(P_1^{loc})^{-1}P_1^{loc}(L_1^2) \cap P_1(P_2)^{-1}(L_2^2) = \tau_1^*(a_1 + a\tau_1^*\tau)\tau_1^*$$

We have e.g.

$$\tau_1 \in (P_1^{loc})^{-1}P_1^{loc}(L_1^2) \cap P_1(P_2)^{-1}(L_2^2) \setminus L_1^2.$$

In the second example we have seen that the mutual normality is not a necessary condition for commutativity between the supremal normal sublanguage and the synchronous product.

6.2 Verification of Mutual Normality

In this subsection we suggest a test for mutual normality. The algorithm for the test we propose will be based on similar algorithms for normality in supervisory control of (monolithic) discrete-event systems. Indeed, it is sufficient to notice that L_1 and L_2 are mutually normal iff L_1 is normal with respect to $P_1(P_2)^{-1}(L_2)$ and P_1^{loc}; and L_2 is normal with respect to $P_2(P_1)^{-1}(L_1)$ and P_2^{loc}. Thus the verification of mutual normality is reduced to the verification of normality of L_1 and L_2 with respect to the plant languages $P_1(P_2)^{-1}(L_2)$ and $P_2(P_1)^{-1}(L_1)$, respectively. There are several tests for checking normality in the DES literature. For example, algorithms in [5], [8] or [9] can be used. Notice that these new plant languages are regular provided L_1 and L_2 are regular. A procedure for construction of recognizers for projected languages and inverse projections of regular languages are known and are discussed in [36]. Let G_1 and G_2 denote the automata recognizing the new plants $P_1(P_2)^{-1}(L_2)$ and $P_2(P_1)^{-1}(L_1)$, respectively, S_1 denotes a recognizer of L_1 and S_2 denotes a recognizer of L_2. We propose the following procedure for checking the mutual normality of L_1 and L_2. The only subtlety is that since L_i are not in general sublanguages of $P_i(P_j)^{-1}(L_j)$, $i, j = 1, 2$, S_i cannot be subautomata of G_i, $i = 1, 2$. Therefore, a generalisation of our concept of normal relation is needed.

Definition 6.3. *(Normal relation.) Let two (partial) automata $S = (S, \langle o_1, t_1 \rangle)$ and $G = (G, \langle o, t \rangle)$ with initial states $s_0 \in S$ and $q_0 \in G$, t_1 denoted by $\rightarrow_1$,and t denoted by $\rightarrow$ are given. A binary relation $N(S,G)$ on $S \times G$ is called a* normal relation *if for any $\langle s, t \rangle \in N(S,G)$ the following items hold:*

(i) $\forall a \in A \ : \ s \xrightarrow{a}_1 s_a$ *and* $t \xrightarrow{a} t_a \quad \Rightarrow \quad \langle s_a, t_a \rangle \in N(S,G)$

(ii) $\forall u \in A_{uo} : \ t \xrightarrow{u} t_u \quad \Rightarrow \quad s \xrightarrow{u}_1 s_u.$

(iii) $\forall a \in A_o : t \xrightarrow{a} t_a$ *and* $(\exists s' : \langle s, s' \rangle \in Aux(S_1) : s' \xrightarrow{a}_1 {s'}_a) \ \Rightarrow \ s \xrightarrow{a}_1 s_a.$

An easy modification of the proof of the corresponding theorem in [17] cited below (Theorem 6.4) shows that

Theorem 6.4. *A (partial) language K is $(L,P)-$normal iff there exists a normal relation $N(S,G)$ on $S \times G$ such that $\langle s_0, q_0 \rangle \in N(S,G)$.*

Denote by $l_2^S(.)$ the second components of the behavior homomorphisms, i.e. for a partial automaton S the behavior of $s_0 \in S$ is given by $l^S(s_0) = (l_1^S(s_0), l_2^S(s_0))$.

Algorithm 2. *(Checking of mutual normality.)*
1. Construct recognizers of $L_1^2 \subseteq A_1^*$ and $L_2^2 \subseteq A_2^*$: partial automata S_1 and S_2 with initial states s_0^1 and s_0^2 such that $l_2^{S_i}(s_0^i) = L_i^2, \ i = 1, 2$. Their transition functions are denoted by $\rightarrow_{S_i}, \ i = 1, 2$.
2. Construct recognizers of $P_1(P_2)^{-1}(L_2)$ and $P_2(P_1)^{-1}(L_1)$: partial automata G_1 and G_2 with initial states q_0^1 and q_0^2 such that $l_2^{G_1}(q_0^1) = P_1(P_2)^{-1}(L_2^2)$ and $l_2^{G_2}(q_0^2) = P_2(P_1)^{-1}(L_1^2)$. Their transition functions are denoted by $\rightarrow_{G_i}, \ i = 1, 2$.
3. Construct the relations $Aux(S_1)$ and $Aux(S_2)$.
4. Construct and check relations $N_i \subseteq S_i \times G_i$ for $i = 1, 2$ using (a) and (b) below:
For i=1 to 2 do
begin
(a) $\langle s_0^i, q_0^i \rangle \in N_i$; $s_i := s_0^i$; $t_i := q_0^i$; $Abort := 0$;
Construct $N_i := \{\langle (s_0^i)_w, (q_0^i)_w \rangle \mid w \in l_2^{S_i}(s_0^i) \cap l_2^{G_i}(q_0^i)\}$
by 'browsing through S_i and G_i' using 'a-transitions' for all $a \in A$:
Add iteratively new $\langle (s_i), (t_i) \rangle \in N_i$
(b) If (ii) or (iii) of Definition 6.3 is violated for $s = s_i$ and $t = t_i$ then $Abort := 1$
If $Abort = 1$ then return "L_1 and L_2 are not mutually normal" and goto 6.
end (of For)
5. If $Abort = 0$ then return "L_1 and L_2 are mutually normal"
6. The end (of algorithm)

The algorithm terminates, because it browses through the transition structure of the automata representations that are finite. We notice that our algorithm for checking mutual normality is of an exponential worst-case complexity, but in the sizes of local automata, which are much smaller than the global automaton, especially if there is a large number of small local components, which case is very frequent in applications.

Roughly speaking, mutual normality means that the closure under taking derivatives of the pairs of initial states must verify conditions (ii)-(iii) of Definition 6.3. There is a canonical way to verify this: whenever a new pair of derivatives is added to the relation, conditions (ii)-(iii) of Definition 6.3 are checked at step 4(b) and either one of them is violated and the procedure aborts ($Abort = 1$) meaning that the mutual normality does not hold or a new pair is added to the relation and we eventualy end up by constructing 2 normal relations proving the mutual normality of L_1 and L_2. In the second case the termination is due to exhaustion of transitions leading from related states in their automata representations and not by violation of conditions of the (generalised) normal relation.

6.3 Computational complexity of monolithic vs. modular computation

In this subsection the computational complexities of monolithic and modular computation of supremal normal sublanguages is discussed. The following symbols will be used.

$$\begin{array}{rcl}
n_m \in \mathbb{N} & & \text{number of modules,} \\
n_i & & \text{size of the minimal state set of a recognizer of module } i \in \mathbb{Z}_{n_m}, \\
n^* & = & \max_{i \in \mathbb{Z}_{n_m}} n_i \in \mathbb{N}, \\
n_L & = & \text{size of the minimal state set of the recognizer} \\
& & \text{of the global plant,} \\
k_i & & \text{size of the minimal state set of a recognizer } \; i \in \mathbb{Z}_{n_m}, \\
& & \text{of the local specification } K_i \subseteq A_i^*, \\
k^* & = & \max_{i \in \mathbb{Z}_{n_m}} k_i \in \mathbb{N}, \\
n_K & & \text{size of the minimal state set of the recognizer} \\
& & \text{of the specification.}
\end{array}$$

We have the following simple inequalities and bounds:

$$n_L \leq \prod_{i=1}^{n_m} n_i \leq (n^*)^{n_m}, \;\; n_K \leq \prod_{i=1}^{n_m} k_i \leq (k^*)^{n_m}$$

The time complexity of the computation of the supremal normal sublanguage is stated in [5] 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 in used the paper [16]. The formulas need to be used to derive an explicit expression for the time complexity. Below is used the following formula

$$O(2^{n_L \cdot n_K}). \tag{3}$$

It follows that the complexity of the monolithic (global) computation is double exponential:

$$O(2^{(n^*)^{n_m} \cdot (k^*)^{n_m}}).$$

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 normal sublanguage takes $O(2^{n^* \cdot k^*})$for local specifications. According to [42] there exists a polynomial time algorithm for checking observability. Its minor modification provides a polynomial time algorithm for checking normality. The verification of mutual normality of local plant anguages is then polynomial in terms of n^* and 2^{n^*}. Note that the term 2^{n^*} 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 these can be calculated much faster (cf. [49]). Still the resulting complexity of modular computation with checking of sufficient conditions is clearly only single exponential in terms of n^*.

7 Conclusion

We have studied modular supervisory control with both fully and partially observed modules in the coalgebraic framework. The conditions for preserving the closed-loop languages have been found for both fully and partially observed modular DES. Moreover conditions for commutativity between supremal controllable sublanguages and the synchronous product have been obtained coagebraicly and similar conditions have been obtained for commutativity between supremal normal sublanguages and the synchronous product. Using a similar single-step auxiliary algorithm for computation of supremal normal and controllable sublanguages, it is possible to obtain similar result for commutativity between supremal normal and controllable sublanguages and the synchronous product. These results are important for feasibility of the optimal supervisory control of large distributed plants, because under the derived conditions control synthesis can be exerted locally.

There are many open problems left for future investigations. For instance, all conditions we have obtained are only sufficient conditions. The question is whether they can be weakened, at least in some special cases that occur in some relevant applications to be found. Another direction of future research is to study the conditions for commutativity between the synchronous product and the closed-loop languages using antipermissive control policy. In this paper the blocking issues have not been considered. It is to be expected that for modular DES with partial observations the conditions for nonblocking are also the same as for modular DES with full observations, however it must still be proven.

An effective procedure for verification of mutual normality has been found. Mutual controllability and mutual normality are likely to be much easier to verify than controllability and normality, because the corresponding plant and specification languages involved are local, thus much smaller.

Acknowledgment

A major part of this research has been carried out at CWI Amsterdam, The Netherlands. Financial support of the EU Esprit LTR Project Control and Computation, ISO-2001-33520 and partial financial support of the Grant GA AV No. B100190609 is gratefully acknowledged. This research was also supported by the Academy of Sciences of the Czech Republic, Institutional Research Plan No. AV0Z10190503.

References

[1] P. Aczel and N. Mendler. A final coalgebra theorem. In D.H. Pitt, D.E. Ryeheard, P. Dybjer, A.M. Pitts, A. Poigne (Eds.), *Proc. Category theory and Computer Science*, Lecture Notes in Computer Science, Volume 389, 1989, 357-365.

[2] G. Barrett and S. Lafortune. Bisimulation, the Supervisory Control Problem and Strong Model Matching for Finite State Machines. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 8:377-429, 1998.

[3] G. Barrett and S. Lafortune. Decentralized Supervisory Control with Communicating Controllers. *IEEE Trans. on Automatic Control*, 45:1620-1638, 2000.

[4] A. Bergeron. On the Rational Behaviors of Concurrent Timers. *Theoretical Computer Science* , 189:229-237, 1995.

[5] 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* 15:111-117, 1990.

[6] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varayia. Supervisory Control of a Class of Discrete Event Processes. *IEEE Trans. Automatic Control*, 33:249-260, 1988.

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

[8] H. Cho and S. I. Marcus. On Supremal Languages of Classes of Sublanguages that Arise in Supervisor Synthesis Problems with Partial Observations.JKumar 00 *Mathematics of Control, Signal, and Systems*, 2:47-69, 1989.

[9] H. Cho and S. I. Marcus. Supremal and Maximal Sublanguages Arising in Supervisor Synthesis Problems with Partial Observations. *Math. Systems Theory*, 22:171-211, 1989.

[10] S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, New York and London, 1974.

[11] S. Eilenberg and J.C. Moore. Adjoint functors and triples. *Illinois J. Math.*, Vol. 9, pp. 381-398, 1965.

[12] M. Fabian and R. Kumar. Mutually Nonblocking Supervisory Control of Discrete Event Systems . *Automatica*, pp. 1863-1869, vol. 36, 2000.

[13] R. Grossman and R. G. Larson. The realization map of input-output maps using bialgebras. *Forum Math.* 4 (1992), 109–121.

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

[15] S. Jiang and R. Kumar. *Decentralized Control of Discrete Event Systems with Specialization to Local Control and Concurrent Systems*. IEEE Transactions on Systems, Man, and Cybertetics- Part B: Cybernetics, Vol. 30, No. 5, pp. 653-660, October 2000.

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

[17] J.Komenda. Computation of Supremal Sublanguages of Supervisory Control Using Coalgebra. Proceedings *WODES'02*, Workshop on Discrete-Event Systems, Zaragoza, p. 26-33, October 2-4, 2002.

[18] J. Komenda. Coalgebra and Supervisory Control of Discrete-Event Systems with Partial Observations. Proceedings of *MTNS 2002*, Notre Dame (IN), August 2002.

[19] Supremal normal sublanguages of large distributed discrete event systems . In Proceedings of WODES 2004, Reims, France, September 2004.

[20] J. Komenda and and 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), 257-315, 2005.

[21] 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, pp. 175-184, Wroclaw, Poland, September 2004.

[22] S. Lafortune and E. Chen. The Infimal Closed and Controllable Superlanguage and its Applications in Supervisory Control, *IEEE Trans. on Automatic Control*, 35:398-405, 1990.

[23] F. Lin and W.M. Wonham. Decentralized Supervisory Control of Discrete-Event Systems, *Information Sciences* 44:199-224, 1988.

[24] F. Lin and W.M. Wonham, Decentralized Control and Coordination of Discrete-Event Systems with Partial Observations, *IEEE Trans. Automatic Control*, 35:1330-1337, 1990.

[25] F. Lin and W.M. Wonham. Verification of Nonblocking in Decentralized Supervision. *Control-Theory and Advanced Technology,* 7:223-232, March 1991.

[26] R. Milner. Communication and Concurrency. *Prentice Hall International Series in Computer Science.* Prentice Hall International, New York, 1989.

[27] A. Overkamp and J.H. van Schuppen. Maximal Solutions in Decentralized Supervisory Control, *SIAM Journal on Control and Optimization*, 39:492-511, 2000.

[28] K. Rohloff and S. Lafortune. The Control and Verification of Similar Agents Operating in a Broadcast Network Environment. *In Proceedings CDC 2003*, Hawaii, USA.

[29] K. Rohloff and S. Lafortune. On the Computational Complexity of the Verification of Modular Discrete-Event Systems. In *Proc. 41 st IEEE Conference on Decision and Control, Las Vegas, Nevada, USA, December 2002.*

[30] J.J.M.M. Rutten. Automata and Coinduction (an Exercise in Coalgebra). *Research Report CWI*, SEN-R9803, Amsterdam, May 1998. Available also at `http://www.cwi.nl/~janr`.

[31] 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`.

[32] J.J.M.M. Rutten. Universal Coalgebra: A Theory of Systems. *Theoretical Computer Science* 249:3-80, 2000.

[33] J.J.M.M. Rutten. Fundamental Study. Behavioural Differential Equations: a Coinductive Calculus of Streams, Automata, and Power Series. *Theoretical Computer Science* 308(1):1-53, 2003.

[34] P.J. Ramadge and W.M. Wonham. The Control of Discrete-Event Systems. *Proc. IEEE*, 77:81-98, 1989.

[35] K. Rudie and W.M. Wonham. The Infimal Prefix-Closed and Observable Superlanguage of a Given Language. *Systems & Control Letters* 15:361-371, 1990.

[36] K. Rudie and W.M. Wonham. Think Globally, Act Locally: Decentralized Supervisory Control, *IEEE Trans. on Automatic Control*, 37:1692-1708, 1992.

[37] K. Rudie and J. Willems. The Computational Complexity of Decentralized Discrete-Event Control Problems, *IEEE Trans. on Automatic Control*, 40:313-1319, 1995.

[38] K. Schmidt, J. Reger, and T. Moor. Hierarchical control of structural decentralized DES. In Proceedings of WODES 2004, Reims, France, September 2004.

[39] E.D. Sontag. Polynomial response maps. *Lecture Notes in Control and Information Sciences*, volume 13, Springer-Verlag, Berlin, 1979.

[40] J.G. Thistle. Supervisory Control of Discrete-Event Systems. *Mathematical and Computer Modeling*, 11/12:25-53, 1996.

[41] J.G. Thistle and W.M. Wonham. Supervision of Infinite Behavior of Discrete-Event Systems, *SIAM Journal on Control and Optimization* 32:1098-1113, 1994.

[42] J.N. Tsitsiklis. On the Control of Discrete-Event Dynamical Systems. *Mathematics of Control, Signal, and Systems*, 95-107, 1989.

[43] Y. Willner and M. Heymann. Supervisory Control of Concurrent Discrete-Event Systems. *International Journal of Control,* 54:1143-1166, 1991.

[44] K.C. Wong and S. Lee. Structural Decentralized Control of Concurrent Discrete-Event Systems. *European Journal of Control*, 8:477-491, 2002.

[45] K.C. Wong and W.M. Wonham. Modular Control and Coordination of Discrete-Event Systems. *Discrete Event Dynamical Systems: Theory and Applications*, 8:247-297, 1998.

[46] W.M. Wonham and P.J. Ramadge. Modular Supervisory Control of Discrete-Event Processes, *Mathematics of Control, Signal and Systems*, 1:13-30, 1988.

[47] T.S. Yoo, S. Lafortune, and F. Lin. A Uniform Approach for Computing Supremal Sublanguages Arising in Supervisory Control theory. *Preprint, Dept. of Electrical Engineering and Computer Science, University of Michigan*, Ann Arbor 2001.

[48] T.S. Yoo, S. Lafortune. General Architecture for Decentralized Supervisory Control of Discrete-Event Systems. *Discrete Event Dynamic Systems: Theory and Applications*, 12:335-377, 2002.

[49] K.C. Wong. On the complexity of projections of discrete-event systems, in Proc. 4th Int. Workshop Discrete Event Syst. (WODES'98), Cagliari, Italy, 1998, pp. 201-206.

A Coalgebra and Coinduction

Final coalgebras give rise to coinductive definition and proof principle. These are heavily used throughout the paper. In this appendix several concepts for control of DES are defined which are used in the paper.

A.1. Coinductive definitions

Recall from [31] the following coinductive definitions of the synchronous and the supervised products with full observations. Large scale DES are frequently formed as parallel compositions of concurrently running components. 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 language over $A_1 \cup A_2$ with the following coinductive definition:

Definition A.1. *(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 $(K \parallel L)\downarrow$ *iff* $K\downarrow$ *and* $L\downarrow$.

Now we recall from [31] the operation of supervised product that represents the closed-loop laguage, where the first language (K) acts as a supervisor (or specification language) and the second language (L) is the open-loop (plant) language. In the following definition $A_u \subseteq A$ denotes the subset of uncontrollable events (those that cannot be prevented from happening by any supervisor).

Definition A.2. *(Supervised product with full observations)*

$$(K/_{A_u}L)_a = \begin{cases} K_a/_{A_u}L_a & \text{if } K \xrightarrow{a} \text{ and } L \xrightarrow{a} \\ 0/_{A_u}L_a & \text{if } K \not\xrightarrow{a} \text{ and } L \xrightarrow{a} \text{ and } a \in A_u \\ \emptyset & \text{otherwise} \end{cases}$$

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

It is proven in [20] that $(K/_{A_u}L)$ equals the infimal controllable superlanguage of K. However, in a typical situation of supervisory control problem safety is the main issue. Therefore supremal controllable sublanguages are more interesting than infimal controllable superlanguages. Now we recall from [20] the following binary operation on partial languages:

Definition A.3. *(Supremal controllable sublanguage) Define the following binary operation on (partial) languages for all $K, L \in \mathcal{L}$ and $\forall a \in A$:*

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

and $(K/_C^S L)\downarrow$ *iff* $L\downarrow$.

One can easily see a very simple intuition behind the Definition A.3. It is nothing but a language formulation of safe (under control) subset of states (i.e. the complement of weakly forbidden set of states). This correspond to subset of states from which it is possible to evolve into forbiden states (equivalent to strings not in K)in an uncontrollable fashion. We have shown in [20] that for a partial order that considers only second (prefix-closed) componets of the languages involved:

Theorem A.4. $(K/_C^S L) = \sup\{M \subseteq K : \; M$ *is controllable with respect to* L *and* $A_u\}$*, i.e.* $K/_C^S L$ *equals the supremal controllable sublanguage of* K.