On Conditional Decomposability

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

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

Abstract

The requirement of a language to be conditionally decomposable is imposed on a specification language in the coordination supervisory control framework of discrete-event systems. In this paper, we present a polynomial-time algorithm for the verification whether a language is conditionally decomposable with respect to given alphabets. Moreover, we also present a polynomial-time algorithm to extend the common alphabet so that the language becomes conditionally decomposable. A relationship of conditional decomposability to nonblockingness of modular discrete-event systems is also discussed in this paper in the general settings. It is shown that conditional decomposability is a weaker condition than nonblockingness.

Keywords: Discrete-event system, coordination control, conditional decomposability.
2000 MSC: 93C65, 93A99, 93B50

1. Introduction

In the Ramadge-Wonham supervisory control framework, discrete-event systems are represented by deterministic finite automata. Given a specification language (usually also represented by a deterministic finite automaton), the aim of supervisory control is to construct a supervisor so that the closed-loop system satisfies the specification [1]. The theory is widely developed for the case where the system (plant) is monolithic. However, large engineering systems are typically constructed compositionally as a collection of many small components (subsystems) that are interconnected by rules; for instance, using a synchronous product or a communication protocol. This is especially true for discrete-event systems, where different local components run in parallel. Moreover, examples of supervisory control of modular discrete-event systems show that a coordinator is often necessary for achieving the required properties because the purely decentralized control architecture may fail in achieving these goals.

The notion of separability of a specification language has been introduced in [2], and says that a language K over an alphabet $\bigcup_{i=1}^{n} E_i$, $n \geq 2$, is separable if $K = \|_{i=1}^{n} P_i(K)$, where for all $i = 1, 2, \ldots, n$, $P_i : (\bigcup E_i)^* \to E_i^*$ is a projection. A specification for a global system is separable if it can be represented (is fully determined) by local specifications for the component subsystems. It is very closely related to the notion of decomposability introduced in [3, 4] for decentralized discrete-event systems, which is also further studied in, e.g., [5]. Decomposability is a slightly more general condition because it involves not only the specification, but also the plant language, that is, a language $K \subseteq L$ over an alphabet $\bigcup_{i=1}^{n} E_i$, $n \geq 2$, is decomposable with respect to a plant language L if $K = \|_{i=1}^{n} P_i(K) \| L$: separability is then decomposability where $L = (\bigcup_{i=1}^{n} E_i)^*$ is the set of all strings over the global alphabet. In this paper, we slightly abuse the terminology and call a separable language in the sense of [2] also decomposable. It has been shown in [2] that decomposability is important because it is computationally cheaper to compute locally synthesized supervisors that constitute a solution of the supervisory control problem for this decomposable specification. Recently, the notion of decomposability has also been extended to automata as an automaton decomposability in, e.g., [6].

---

*Corresponding author. Institute of Mathematics, Academy of Sciences of the Czech Republic, Žižkova 22, 616 62 Brno, Czech Republic, Tel. +420222090784, Fax. +420541218657

Email addresses: `komenda@ipm.cz` (Jan Komenda), `masopust@math.cas.cz` (Tomáš Masopust), `J.H.van.Schuppen@cwi.nl` (Jan H. van Schuppen)

However, the assumption that a specification language is decomposable is too restrictive. Therefore, several authors have tried to find alternative techniques for general indecomposable specification languages; for instance, the approach of [7] based on partial controllability, which requires that all shared events are controllable, or the shared events must have the same controllability status (but then an additional condition of so-called mutual controllability [8] is needed).

In this paper, we study a weaker version of decomposability, so-called *conditional decomposability*, which has recently been introduced in [9] and studied in [10, 11] in the context of coordination supervisory control of discrete-event systems. It is defined as decomposability with respect to local alphabets augmented by the coordinator alphabet. The word conditional means that although a language is not decomposable with respect to the original local alphabets, it becomes decomposable with respect to the augmented ones, i.e., decomposability is only guaranteed (conditioned) by local event set extensions by coordinator events.

In the coordination control approach of modular discrete-event systems, the plant is formed as a parallel composition of two or more subsystems, while the specification language is represented over the global alphabet. Therefore, the property of conditional decomposability is required in this approach to distribute parts of the specification to the corresponding components to solve the problem locally. More specifically, we need to ensure that there exists a corresponding part of the specification for the coordinator and for each subsystem composed with the coordinator. Thus, if the specification is conditionally decomposable, we can take this decomposition as the corresponding parts for the subsystems composed with a coordinator and solve the problem locally.

Conditional decomposability depends on the alphabet of the coordinator, which can always be extended so that the specification is conditionally decomposable. In the worst (but unlikely) case all events must be put into the coordinator alphabet to make a language conditionally decomposable. But in the case when the coordinator alphabet would be too large it is better to divide the local subsystems into groups that are only loosely coupled and introduce several coordinators on smaller alphabets. In this paper, a polynomial-time algorithm is provided for the verification whether a language is conditionally decomposable. We make an important observation that the algorithm is linear in the number of local alphabets, while algorithms for checking similar properties (such as decomposability and coobservability) suffer from the exponential-time complexity with respect to the number of local alphabets. This algorithm is then modified so that it extends the coordinator alphabet to make the specification language conditionally decomposable. Furthermore, we discuss a relationship of conditional decomposability to nonblockingness of a coordinated system, where a coordinated system is understood as a modular system composed of two or more subsystems and a coordinator.

Finally, since one of the central notions of this paper is the notion of a (natural) projection, the reader is referred to [12] for more information on the state complexity of projected regular languages.

The rest of this paper is organized as follows. In Section 2, basic definitions and concepts of automata theory and discrete-event systems are recalled. In Section 3, a polynomial-time algorithm for testing conditional decomposability for a general monolithic system is presented. In Section 4, this algorithm is modified to extend the coordinator alphabet so that the specification becomes conditionally decomposable. In Section 5, the relation of nonblockingness of a coordinated system with conditional decomposability is discussed. The conclusion with hints for future developments is presented in Section 6.

2. Preliminaries and definitions

In this paper, we assume that the reader is familiar with the basic concepts of supervisory control theory [13] and automata theory [14]. For an alphabet E, defined as a finite nonempty set, E^* denotes the free monoid generated by E, where the unit of E^*, the empty string, is denoted by ε. A *language* over E is a subset of E^*. A prefix closure $\overline{L}$ of a language $L \subseteq E^*$ is the set of all prefixes of all words of L, i.e., it is defined as the set $\overline{L} = \{w \in E^* \mid \exists u \in E^* : wu \in L\}$. A language L is said to be prefix-closed if $L = \overline{L}$.

In this paper, the notion of a generator is used to denote an incomplete deterministic finite automaton. A *generator* is a quintuple $G = (Q, E, \delta, q_0, F)$, where Q is a finite set of *states*, E is an *input alphabet*, $\delta : Q \times E \to Q$ is a *partial transition function*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final or marked states*. In the usual way, δ is inductively extended to a function from $Q \times E^*$ to Q. The language *generated* by G is defined as the set $L(G) = \{w \in E^* \mid \delta(q_0, w) \in Q\}$, and the language *marked* by G is defined as the set $L_m(G) = \{w \in E^* \mid \delta(q_0, w) \in F\}$. Moreover, we use the predicate $\delta(q, a)!$ to denote that the transition $\delta(q, a)$ is defined in state $q \in Q$ for event $a \in E$.

For a generator G, let $\texttt{trim}(G)$ denote the *trim* of G, that is, a generator $\texttt{trim}(G)$ such that $\overline{L_m(\texttt{trim}(G))} = L(\texttt{trim}(G)) = \overline{L_m(G)}$. In other words, all reachable states of G from which no marked state is reachable are removed (including the corresponding transitions), and only reachable states are considered in $\texttt{trim}(G)$, see [13, 15]. A generator G is said to be *nonblocking* if $\overline{L_m(G)} = L(G)$. Thus, $\texttt{trim}(G)$ is always nonblocking.

A *(natural) projection* $P : E^* \to E_0^*$, where $E_0 \subseteq E$ are alphabets, is a homomorphism defined so that $P(a) = \varepsilon$, for $a \in E \setminus E_0$, and $P(a) = a$, for $a \in E_0$. The *inverse image* of the projection P, denoted by $P^{-1} : E_0^* \to 2^{E^*}$, is defined so that for a language L over the alphabet E_0, the set $P^{-1}(L) = \{s \in E^* \mid P(s) \in L\}$. In what follows, we use the notation P_j^i to denote the projection from E_i to E_j, that is, $P_j^i : E_i^* \to E_j^*$. In addition, we use the notation $E_{i+j} = E_i \cup E_j$, and, thus, P_k^{i+j} denotes the projection from E_{i+j} to E_k. If the projection is from the union of all the alphabets, then we simply use the notation $P_i : (\bigcup_j E_j)^* \to E_i^*$.

Let $L_1 \subseteq E_1^*$ and $L_2 \subseteq E_2^*$ be two languages. The *parallel composition of L_1 and L_2* is defined as the language

$$L_1 \parallel L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2),$$

where $P_1 : (E_1 \cup E_2)^* \to E_1^*$ and $P_2 : (E_1 \cup E_2)^* \to E_2^*$. A similar definition in terms of generators follows. Let $G_1 = (X_1, E_1, \delta_1, x_{01}, F_1)$ and $G_2 = (X_2, E_2, \delta_2, x_{02}, F_2)$ be two generators. The *parallel composition of G_1 and G_2* is the generator $G_1 \parallel G_2$ defined as the accessible part of the generator $(X_1 \times X_2, E_1 \cup E_2, \delta, (x_{01}, x_{02}), F_1 \times F_2)$, where

$$\delta((x, y), e) = \begin{cases} (\delta_1(x, e), \delta_2(y, e)), & \text{if } \delta_1(x, e)! \text{ and } \delta_2(y, e)!; \\ (\delta_1(x, e), y), & \text{if } \delta_1(x, e)! \text{ and } e \notin E_2; \\ (x, \delta_2(y, e)), & \text{if } e \notin E_1 \text{ and } \delta_2(y, e)!; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

The automata definition is related to the language definition by the following properties: $L(G_1 \parallel G_2) = L(G_1) \parallel L(G_2)$ and $L_m(G_1 \parallel G_2) = L_m(G_1) \parallel L_m(G_2)$, see [13].

The automata-theoretic concept of nonblockingness of a composition of two generators G_1 and G_2 is equivalent to the language-theoretic concept of nonconflictness of two languages $L_m(G_1)$ and $L_m(G_2)$ if the generators G_1 and G_2 are nonblocking. Recall that two languages L_1 and L_2 are *nonconflicting* if $\overline{L_1} \parallel \overline{L_2} = \overline{L_1 \parallel L_2}$, cf. [15, 16, 17].

Let G be a generator and P be a projection, then $P(G)$ denotes the minimal generator such that $L_m(P(G)) = P(L_m(G))$ and $L(P(G)) = P(L(G))$. For a construction of $P(G)$, the reader is referred to [13, 15].

Now, the main concept of interest of this paper, the concept of conditional decomposability, is defined. See also [9, 10, 11, 18] for the applications and further discussion concerning this concept.

Definition 1 (Conditional decomposability). *A language K over an alphabet $E_1 \cup E_2 \cup \ldots \cup E_n$, $n \geq 2$, is said to be* conditionally decomposable *with respect $E_1, E_2, \ldots, E_n$, and E_k, where $\bigcup_{i,j \in \{1,2,\ldots,n\}}^{i \neq j} (E_i \cap E_j) \subseteq E_k \subseteq \bigcup_{j=1}^n E_j$, if*

$$K = P_{1+k}(K) \parallel P_{2+k}(K) \parallel \ldots \parallel P_{n+k}(K).$$

Recall that P_{i+k} denotes the projection from $\bigcup_{j=1}^n E_j$ to E_{i+k}.

Note that $\|_{i=1}^n P_{i+k}(K) = (\|_{i=1}^n P_{i+k}(K)) \parallel P_k(K)$ because $P_{i+k}(K) \subseteq (P_k^{i+k})^{-1} P_k(K)$, which follows from the fact that $P_k^{i+k} P_{i+k}(K) = P_k(K)$. Hence, $\|_{i=1}^n P_{i+k}(K) \subseteq P_k^{-1} P_k(K)$. Moreover, if the language K is given as a parallel composition of n languages (over the required alphabets), then it is conditionally decomposable.

Lemma 2. *A language $K \subseteq (E_1 \cup E_2 \cup \ldots \cup E_n)^*$ is conditionally decomposable with respect to alphabets $E_1, E_2, \ldots, E_n, E_k$ if and only if there exist languages $M_{i+k} \subseteq E_{i+k}^*$, $i = 1, 2, \ldots, n$, such that $K = \|_{i=1}^n M_{i+k}$.*

PROOF. If $K = \|_{i=1}^n P_{2+k}(K)$, define $M_{i+k} = P_{i+k}(K)$, for $i = 1, 2, \ldots, n$. On the other hand, assume that there exist languages $M_{i+k} \subseteq E_{i+k}^*$, $i = 1, 2, \ldots, n$, such that $K = \|_{i=1}^n M_{i+k}$. Obviously, $P_{i+k}(K) \subseteq M_{i+k}$, $i = 1, 2, \ldots, n$, which implies that $\|_{i=1}^n P_{i+k}(K) \subseteq K$. As it always holds that $K \subseteq P_{i+k}^{-1}[P_{i+k}(K)]$, the definition of the synchronous product implies that $K \subseteq \|_{i=1}^n P_{i+k}(K)$. $\qquad\square$

Note that $K = \|_{i=1}^n M_{i+k}$ implies that the languages $P_{i+k}(K) \subseteq M_{i+k}$, for $i = 1, 2, \ldots, n$, which means that $P_{i+k}(K)$ are the smallest languages whose parallel composition results in K. In other words, if K is conditionally decomposable, then $P_{i+k}(K)$, $i = 1, 2, \ldots, n$, is the smallest decomposition of K with respect to the corresponding alphabets.

3

3. Polynomial Test of Conditional Decomposability

In this section, we first construct a polynomial-time algorithm for the verification of conditional decomposability for alphabets E_1, E_2, and E_k, that is, for the case $n = 2$, and then show how this is used to verify conditional decomposability for a general $n \geq 2$. To this end, consider a language L over $E_1 \cup E_2$, marked by a generator G. To verify whether or not L is conditionally decomposable with respect E_1, E_2, and E_k, we construct a new structure as a parallel composition of two copies of G, denoted $f_{i+k}(G)$, for $i = 1, 2$, (see Example 3 and Figure 2) that simultaneously verifies that each word of $P_{1+k}(L) \parallel P_{2+k}(L)$ also belongs to $L = L_m(G)$; $f_{i+k}(G)$ is constructed from the generator G by renaming each event $e \in E_{j-k} = E_j \setminus E_k$, $j \neq i$, by a new event $\tilde{e} \in \tilde{E}_{j-k}$. In other words, each event e which is not observed by G according to the observable alphabet $E_i \cup E_k$ is replaced with a new event. Thus, the copy $f_{i+k}(G)$ is over the alphabet $E_{i+k} \cup \tilde{E}_{j-k}$, as demonstrated in the following example.

Example 3. *Consider the language $L_m(G)$ marked by the generator G depicted in Figure 1(a), where the corresponding alphabets are $E_1 = \{a, b, d\}$, $E_2 = \{a, c, d\}$, and $E_k = \{a, d\}$. The isomorphic generators $f_{1+k}(G)$ with renamed event c, and $f_{2+k}(G)$ with renamed event b are depicted in Figure 1(b) and Figure 1(c), respectively.* ◇

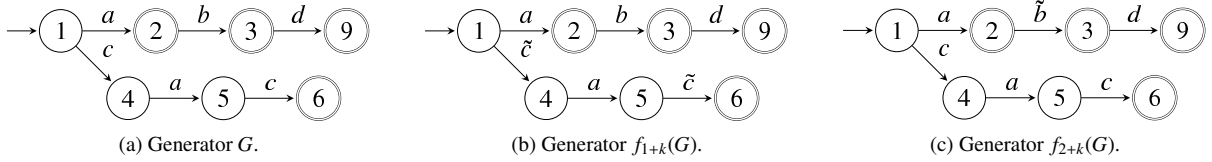(a) Generator G.   (b) Generator $f_{1+k}(G)$.   (c) Generator $f_{2+k}(G)$.

Figure 1: Generators G, $f_{1+k}(G)$ and $f_{2+k}(G)$.

More specifically, let E_1, E_2, E_k be alphabets such that $E_1 \cap E_2 \subseteq E_k \subseteq E_1 \cup E_2$, and define the global alphabet $E = E_1 \cup E_2$. The structure is constructed as follows:

1. For the alphabet $E_i \setminus E_k$, where $i = 1, 2$, introduce a new alphabet $\tilde{E}_{i-k} = \{\tilde{a} \mid a \in E_i \setminus E_k\}$ that for each event $a \in E_i \setminus E_k$ contains a new event $\tilde{a}$. That is, $\tilde{E}_{i-k} \cap (E_i \setminus E_k) = \emptyset$ and there exists a bijection g_{i-k} from $(E_i \setminus E_k)$ to $\tilde{E}_{i-k}$ such that $g_{i-k}(a) = \tilde{a}$. Note that $\tilde{E}_{1-k} \cap \tilde{E}_{2-k} = \emptyset$ because $E_1 \cap E_2 \subseteq E_k$.

2. Recall that $E_{i+k} = E_i \cup E_k$, for $i = 1, 2$, and let $\tilde{P} : (E \cup \tilde{E}_{1-k} \cup \tilde{E}_{2-k})^* \to E^*$ be a projection.

3. Define two isomorphisms $f_{i+k} : E^* \to (E_{i+k} \cup \tilde{E}_{j-k})^*$, where $i, j \in \{1, 2\}$, $i \neq j$, so that

$$f_{i+k}(a) = \begin{cases} a, & \text{for } a \in E_{i+k}; \\ \tilde{a}, & \text{for } a \in E_j \setminus E_k. \end{cases}$$

Note that it immediately follows that $\tilde{P}(f_{i+k}(L_m(G))) = P_{i+k}(L_m(G))$ because both projections remove all events that are not in E_{i+k}.

4. For a generator $G = (Q, E, \delta, q_0, F)$, we abuse the notation and denote by $f_{i+k}(G) = (Q, E_{i+k} \cup \tilde{E}_{j-k}, \tilde{\delta}, q_0, F)$, where $j \neq i$, the generator isomorphic with G where events are renamed according to the isomorphism f_{i+k}, and the transition function $\tilde{\delta}$ is define as $\tilde{\delta}(q, f_{i+k}(a)) = \delta(q, a)$.

5. Let $L \subseteq E^*$ be a language generated by a minimal generator G, and define the generator

$$\tilde{G} = f_{1+k}(G) \parallel f_{2+k}(G)$$

over the alphabet $E \cup \tilde{E}_{1-k} \cup \tilde{E}_{2-k}$. By the definition of $\tilde{G}$, the assumption that $E_1 \cap E_2 \subseteq E_k$ which ensures that $\tilde{P}$ distributes over the synchronous product (see Lemma 4 below), and Step 3 above, respectively, we have that

$$\begin{aligned} \tilde{P}(L_m(\tilde{G})) &= \tilde{P}(f_{1+k}(L_m(G)) \parallel f_{2+k}(L_m(G))) \\ &= \tilde{P}(f_{1+k}(L_m(G))) \parallel \tilde{P}(f_{2+k}(L_m(G))) \\ &= P_{1+k}(L_m(G)) \parallel P_{2+k}(L_m(G)). \end{aligned} \tag{1}$$

4

Lemma 4 ([15]). *Let $E_1 \cap E_2 \subseteq E_k \subseteq E_1 \cup E_2$, and let $L_1 \subseteq E_1^*$ and $L_2 \subseteq E_2^*$ be languages. Let $P_k : E^* \to E_k^*$ be a projection, then $P_k(L_1 \parallel L_2) = P_k(L_1) \parallel P_k(L_2)$.*

From the equations of (1), we immediately have the following result for conditional decomposability.

Theorem 5. *The language $L_m(G)$ is conditionally decomposable with respect to alphabets E_1, E_2, E_k if and only if it holds that $\tilde{P}(L_m(\tilde{G})) = L_m(G)$.*

PROOF. The proof follows immediately from the definition of conditional decomposability and (1). $\qquad\square$

However, the inclusion $L_m(G) \subseteq P_{1+k}(L_m(G))) \parallel P_{2+k}(L_m(G))) = \tilde{P}(L_m(\tilde{G}))$ always holds. Thus, only the opposite inclusion is of interest. This inclusion, $\tilde{P}(L_m(\tilde{G})) \subseteq L_m(G)$, holds if and only if $L_m(\tilde{G}) \subseteq \tilde{P}^{-1}(L_m(G))$, which gives the following key theorem for testing conditional decomposability.

Theorem 6. *The language $L_m(G)$ is conditionally decomposable with respect to alphabets E_1, E_2, E_k if and only if the inclusion $L_m(\tilde{G}) \subseteq \tilde{P}^{-1}(L_m(G))$ holds.*

PROOF. It remains to prove that $\tilde{P}(L_m(\tilde{G})) \subseteq L_m(G)$ if and only if $L_m(\tilde{G}) \subseteq \tilde{P}^{-1}(L_m(G))$. However, if $\tilde{P}(L_m(\tilde{G})) \subseteq L_m(G)$, then $L_m(\tilde{G}) \subseteq \tilde{P}^{-1}\tilde{P}(L_m(\tilde{G})) \subseteq \tilde{P}^{-1}(L_m(G))$. On the other hand, assume that $L_m(\tilde{G}) \subseteq \tilde{P}^{-1}(L_m(G))$. Then, $\tilde{P}(L_m(\tilde{G})) \subseteq \tilde{P}\tilde{P}^{-1}(L_m(G)) = L_m(G)$. $\qquad\square$

The verification of this inclusion results in Algorithm 1 for checking conditional decomposability of two components in polynomial time. Let a language L be represented by the minimal generator $G = (Q, E, \delta, q_0, F)$ with the complete (total) transition function δ such that $L_m(G) = L$. If the transition function is not complete, the generator can be completed in time $O(|E| \cdot |Q|)$ by adding no more than one non-marked state and the missing transitions. Assume that the alphabets E_1, E_2, and E_k are such that $E_1 \cap E_2 \subseteq E_k \subseteq E_1 \cup E_2 = E$, and see Algorithm 1. To determine

---

Algorithm 1 Conditional decomposability checking.

1: **function** IsCD(G, E_1, E_2, E_k)
2:     Compute $\tilde{G}$ ▷ $O(|E| \cdot |Q|^2)$.
3:     Compute $\tilde{P}^{-1}(L_m(G))$ ▷ $O(|E| \cdot |Q|)$.
4:     Compute complement $co(\tilde{P}^{-1}(L_m(G)))$ ▷ $O(|Q|)$.
5:     **if** $co(\tilde{P}^{-1}(L_m(G))) \cap L_m(\tilde{G}) = \emptyset$ **then**   ▷ $O(|E| \cdot |Q|^3)$.
6:         **return** $L_m(G)$ is CD.
7:     **else**
8:         **return** $L_m(G)$ is not CD.
9:     **end if**
10: **end function**

---

the time complexity of the algorithm, note that the computation is dominated by step 5, and thus the overall time complexity can be stated as $O(|E| \cdot |Q|^3)$. This also means that the space complexity is polynomial with respect to the number of states of the input generator G because we do not need to use more space than $O(|E| \cdot |Q|^3)$. The complexity of individual steps of the algorithm are computed as follows. Step 2 is a parallel composition of two copies of G, which requires to create up to $|Q|^2$ states of the generator $\tilde{G}$, and for each of these states up to $|E|$ transitions. Step 3 requires up to $|E| \cdot |Q|$ steps because in each state, we have to add self-loops labeled by the new symbols from $\tilde{E}_{1-k} \cup \tilde{E}_{2-k}$. The complement in Step 4 is computed by interchanging the marking of states, cf. [19]. That is, marked states are unmarked and vice versa. As G is complete, this results in a generator for the complement. Note that Steps 3 and 4 can be done at the same time. Finally, to decide the emptiness in Step 5 requires up to $|Q|^2 \cdot |Q|$ using a standard product automaton, see [19], where for each state, up to $|E|$ transitions are constructed, and is verified by the reachability of a final state by the depth-first-search procedure in linear time [20]. Note also that it is a longstanding open problem whether the emptiness of intersection of two regular languages generated by generators with m_1 and m_2 states, respectively, can be decided in time $o(m_1 \cdot m_2)$, cf. [21]. If this is possible, then the complexity of our algorithm can be improved accordingly.

We demonstrate our approach in the following example.

Example 7. *Consider the language $L_m(G)$ marked by the generator G depicted in Figure 1(a), where the corresponding alphabets are $E_1 = \{a, b, d\}$, $E_2 = \{a, c, d\}$, and $E_k = \{a, d\}$. The isomorphic generators $f_{1+k}(G)$ with renamed event c, and $f_{2+k}(G)$ with renamed event b are depicted in Figure 1(b) and Figure 1(c), respectively. Their parallel composition $\tilde{G}$ is shown in Figure 2. It is obvious that the string "cacb" belongs to the language $L_m(\tilde{G})$, whereas it does not belong to the language $\tilde{P}^{-1}(L_m(G))$. Thus, by Theorem 6, the language $L_m(G)$ is not conditionally decomposable with respect to alphabets E_1, E_2, E_k.* ◇
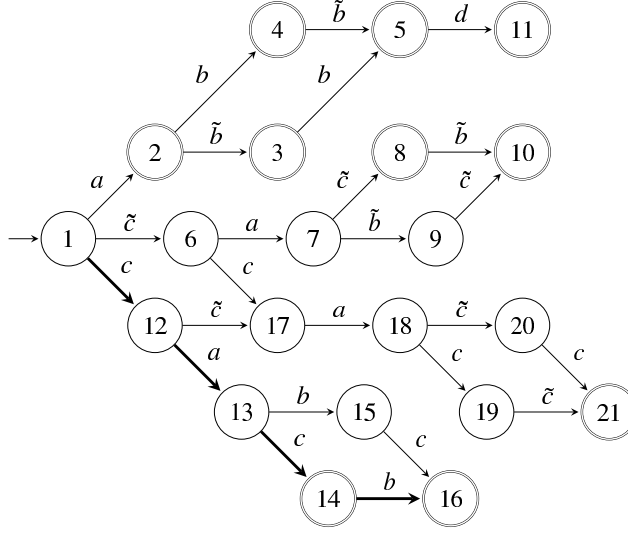

Figure 2: Generator $\tilde{G} = f_{1+k}(G) \parallel f_{2+k}(G)$ with a highlighted word violating conditional decomposability of the language $L_m(G)$.

Now, we generalize this approach to verifying conditional decomposability for a general number of $n \geq 2$ alphabets. The following theorem proves that we can directly use Algorithm 1.

Theorem 8. *Let K be a language, and let E_i, for $i = 1, 2, \ldots, n$, $n \geq 2$, and E_k be alphabets such that $\bigcup_{i \neq j}(E_i \cap E_j) \subseteq E_k \subseteq \bigcup_{j=1}^{n} E_j$. Then, $P_{i+k}(K) \parallel P_{1+2+\ldots+(i-1)+(i+1)+\ldots+n+k}(K) \subseteq K$, for all $i = 1, 2, \ldots, n$, if and only if K is conditionally decomposable with respect to alphabets E_i, $i = 1, 2, \ldots, n$, and E_k.*

PROOF. First, $P_{1+2+\ldots+(i-1)+(i+1)+\ldots+n+k}(K) \subseteq P_{1+k}(K) \parallel P_{2+k}(K) \parallel \ldots \parallel P_{(i-1)+k}(K) \parallel P_{(i+1)+k}(K) \parallel \ldots \parallel P_{n+k}(K)$ because for all $i \in \{1, 2, \ldots, n\}$, we have $P_{i+k}(P_{1+2+\ldots+(i-1)+(i+1)+\ldots+n+k}(K)) = P_{i+k}(K)$. Thus, if K is conditionally decomposable, then $P_{i+k}(K) \parallel P_{1+2+\ldots+(i-1)+(i+1)+\ldots+n+k}(K) \subseteq P_{i+k}(K) \parallel P_{1+k}(K) \parallel \ldots \parallel P_{(i-1)+k}(K) \parallel P_{(i+1)+k}(K) \parallel \ldots \parallel P_{n+k}(K) = K$, for all $i = 1, 2, \ldots, n$.

To prove the opposite implication, assume that K is not conditionally decomposable. Then there exist $t_i = P_{i+k}(w_i)$, for some $w_i \in K$ and for all $i = 1, 2, \ldots, n$, such that $t_1 \parallel t_2 \parallel \ldots \parallel t_n \nsubseteq K$. We prove by induction on $i = 1, 2, \ldots, n-1$ that

$$\{t_i\} \parallel \{t_{i-1}\} \parallel \ldots \parallel \{t_2\} \parallel \{t_1\} \parallel P_{(i+1)+(i+2)+\ldots+n+k}(w_n) \subseteq K. \tag{2}$$

For $i = 1$ and by the assumption, $\{t_1\} \parallel P_{2+3+\ldots+n+k}(w_n) \subseteq P_{1+k}(K) \parallel P_{2+3+\ldots+n+k}(K) \subseteq K$. Thus, we assume that it holds for all $i = 1, 2, \ldots, \ell$, $\ell < n-1$, and we prove it for $i = \ell + 1$. By the induction hypothesis, $\{t_\ell\} \parallel \{t_{\ell-1}\} \parallel \ldots \parallel \{t_2\} \parallel \{t_1\} \parallel P_{(\ell+1)+(\ell+2)+\ldots+n+k}(w_n) \subseteq K$. Then, using the projection $P_{1+2+\ldots+\ell+(\ell+2)+\ldots+n+k}$, we get that

$$P_{1+2+\ldots+\ell+(\ell+2)+\ldots+n+k}\Big(\{t_\ell\} \parallel \{t_{\ell-1}\} \parallel \ldots \parallel \{t_2\} \parallel \{t_1\} \parallel P_{(\ell+1)+(\ell+2)+\ldots+n+k}(w_n)\Big) \subseteq P_{1+2+\ldots+\ell+(\ell+2)+\ldots+n+k}(K)$$

and, by Lemma 4, we get that $P_{1+2+\ldots+\ell+(\ell+2)+\ldots+n+k}\big(\{t_\ell\} \parallel \{t_{\ell-1}\} \parallel \ldots \parallel \{t_2\} \parallel \{t_1\} \parallel P_{(\ell+1)+(\ell+2)+\ldots+n+k}(w_n)\big) = \{t_\ell\} \parallel \{t_{\ell-1}\} \parallel \ldots \parallel \{t_2\} \parallel \{t_1\} \parallel P_{(\ell+2)+\ldots+n+k}(w_n)$. By this equality and the assumption for $i = \ell + 1$, we have

$$\{t_{\ell+1}\} \parallel \Big[\{t_\ell\} \parallel \{t_{\ell-1}\} \parallel \ldots \parallel \{t_2\} \parallel \{t_1\} \parallel P_{(\ell+2)+\ldots+n+k}(w_n)\Big]$$
$$\subseteq P_{(\ell+1)+k}(K) \parallel P_{1+2+\ldots+\ell+(\ell+2)+\ldots+n+k}(K)$$
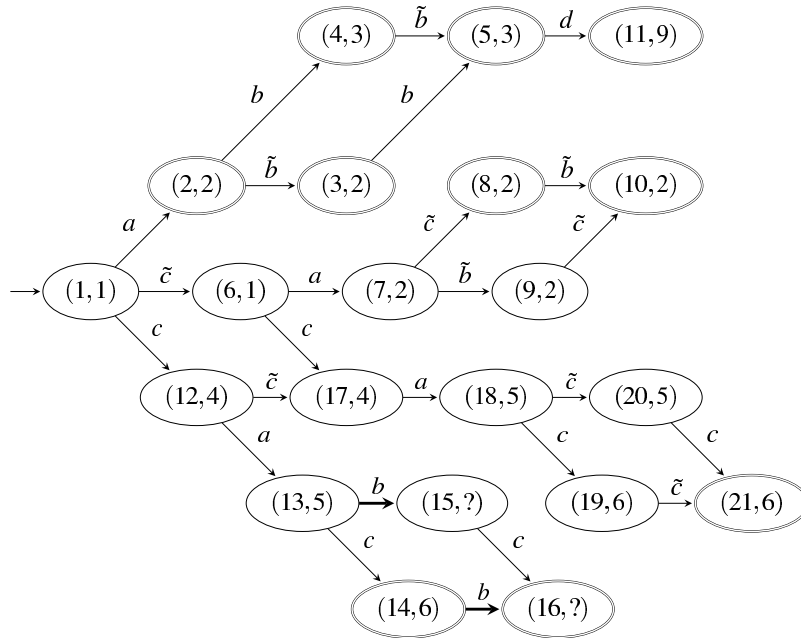$$\subseteq K$$

as claimed. Then, substituting $i = n-1$ to (2), we immediately have that $\{t_{n-1}\} \parallel \{t_{n-2}\} \parallel \ldots \parallel \{t_2\} \parallel \{t_1\} \parallel P_{n+k}(w_n) \subseteq K$, which together with $P_{n+k}(w_n) = t_n$ implies that $\{t_{n-1}\} \parallel \{t_{n-2}\} \parallel \ldots \parallel \{t_2\} \parallel \{t_1\} \parallel \{t_n\} \subseteq K$, which is a contradiction. Thus, K is conditionally decomposable. $\qquad\square$

The previous theorem says that we can check conditional decomposability of a language K by n executions of Algorithm 1. This means that the overall complexity of verifying conditional decomposability for a general number of alphabets, $n \geq 2$, is $O(n \cdot |E| \cdot |Q|^3)$, which is polynomial with respect to the number of states and the number of components.

To conclude this section, note that an example of an r-state automaton with $|E| = 4$ and a projection reaching the exponential upper bound on the number of states, more precisely the upper bound $3 \cdot 2^{r-2} - 1$, has been shown in [22]. Thus, the approach following the definition of conditional decomposability computing projections and parallel composition is exponential for that language even for the case of two alphabets. In comparison, the complexity of our algorithm is polynomial. A preliminary version of this algorithm has been implemented in libFAUDES [23].

4. Extension of the coordinator alphabet

According to Theorem 8, we can again consider only the case $n = 2$. To compute an extension of E_k so that the language becomes conditionally decomposable, we modify Algorithm 1 to Algorithm 2, which uses more structural properties of the structure $\tilde{G}$. First, however, we explain the technique on an example.

Figure 3: Generator $\tilde{G}$ with the corresponding states of $G \times \tilde{G}$. Note that transitions $\delta(5, b)$ and $\delta(6, b)$ are not defined in G, and, therefore, they violate conditional decomposability of the language $L_m(G)$.

Example 9. *Consider the generator G and $\tilde{G}$ of Examples 3 and 7. The main idea of this technique is to construct, step-by-step, the parallel composition of G and $\tilde{G}$, and to verify that all the steps possible in $\tilde{G}$ are also possible in G. In Figure 3, $\tilde{G}$ is extended with the states of G, written in the states of $\tilde{G}$. Note that after reading the string ca, the generator $\tilde{G}$ is in a state from which b can be read, but G being in state 5 can read only c. Because of this symbol b, the language $L_m(G)$ is not conditionally decomposable. The reader can verify that adding b to E_k results in the situation where $L_m(G)$ is conditionally decomposable with respect to E_1, E_2, and $E_k \cup \{b\}$.* ◇

Let a language L be represented by the minimal generator $G = (Q, E, \delta, q_0, F)$ with the total transition function δ such that $L_m(G) = L$. Assume that alphabets E_1, E_2, E_k satisfy $E_1 \cap E_2 \subseteq E_k \subseteq E_1 \cup E_2 = E$, and see Algorithm 2. To prove that the algorithm is correct, note that it computes $L_m(\tilde{G}) \cap \tilde{P}^{-1}(L_m(G))$ because $L_m(\tilde{G}) = L_m(\text{trim}(\tilde{G}))$. If

---

Algorithm 2 Extension of E_k.

1: **procedure** EXTENSION(G, E_1, E_2, E_k)
2:      Compute $\tilde{G}$
3:      Compute $\text{trim}(\tilde{G})$ ▷ Now, we compute, step-by-step, the generator H for $\text{trim}(\tilde{G}) \parallel G$.
4:      Set $Q_H = \{((q_{0,1}, q_{0,2}), q_0)\}$, a pair of initial states of $\tilde{G}$ and G ▷ The initial state of H.
5:      **for all** $((q_1, q_2), q) \in Q_H$ **do**
6:          **for all** $a \in E \cup \tilde{E}_{1-k} \cup \tilde{E}_{2-k}$ **do**
7:              **if** $a \in \tilde{E}_{1-k} \cup \tilde{E}_{2-k}$ and $\delta_{\tilde{G}}((q_1, q_2), a)!$ **then**
8:                  $\delta_H(((q_1, q_2), q), a) = (\delta_{\tilde{G}}((q_1, q_2), a), q)$
9:              **end if**
10:             **if** $a \in E$ and $\delta_{\tilde{G}}((q_1, q_2), a)!$ **then**
11:                 **if** $\delta(q, a)!$ **then**
12:                     $\delta_H(((q_1, q_2), q), a) = (\delta_{\tilde{G}}((q_1, q_2), a), \delta(q, a))$
13:                 **else**
14:                     $E_k = E_k \cup \{a\}$ ▷ Event a is allowed in $\tilde{G}$, but not in G.
15:   ▷ Adding it to E_k solves this problem.
16:                     Restart the procedure with the updated set E_k.
17:                 **end if**
18:             **end if**
19:         **end for**
20:     **end for**
21:     **return** E_k.
22: **end procedure**

---

the condition on line 11 is always satisfied, it means that $L_m(\tilde{G}) \cap \tilde{P}^{-1}(L_m(G)) = L_m(\tilde{G})$. In other words, $L_m(\tilde{G}) \subseteq \tilde{P}^{-1}(L_m(G))$, which means by Theorem 6 that $L_m(G)$ is conditionally decomposable. On the other hand, if the condition on line 11 is not satisfied, there exists a string $s \in L(\text{trim}(\tilde{G})) = \overline{L_m(\tilde{G})}$ such that $\tilde{P}(s) \notin L(G) = \overline{L_m(G)}$, where the last equality follows from the assumption that G is minimal. This implies that $\tilde{P}(L_m(\tilde{G})) \nsubseteq L_m(G)$, hence $L_m(G)$ is not conditionally decomposable by Theorem 5. The algorithms halts because we have only a finite number of events to be added to E_k, and the language is conditionally decomposable for $E_k = E_1 \cup E_2$.

The complexity of this algorithm is $O(|E|^2 \cdot |Q|^3)$, which follows from the complexity of Algorithm 1 and the fact that, in the worst-case, we have to run the algorithm $|E|$ times. Note that the resulting extension depends on the order the states of G and $\tilde{G}$ are examined. It should be clear that, in general, there might be different extensions (with respect to set inclusion) that correspond to different orders. This is a typical issue with algorithms extending the event sets in such a way that a particular property becomes true. There are examples where the algorithm does not construct the minimal possible extension. Therefore, the fundamental question is what is the complexity of constructing the minimal extension of E_k?

5. Relationship of nonblockingness of coordinated systems to conditional decomposability

In this section, we study the relation between conditional decomposability and nonblockingness of coordinated discrete-event systems. A coordinated modular discrete-event system is a system composed (by parallel composition) of two or more subsystems. In this section, we consider the case of one central coordinator. Let $n \geq 2$, and let G_i, $i = 1, 2, \ldots, n$, be generators over the respective alphabets E_i, $i = 1, 2, \ldots, n$. The coordinated system G is defined as $G = G_1 \parallel G_2 \parallel \ldots \parallel G_n \parallel G_k$, where G_k is the coordinator over an alphabet E_k, which contains all shared events; namely, $E_s \subseteq E_k$, where E_s is the set of all events that are shared by two or more components, defined as

$$E_s = \bigcup_{i,j\in\{1,\ldots,n\}}^{i \neq j} (E_i \cap E_j) \,.$$

This is a standard assumption in hierarchical decentralized control where the coordinator level plays a role of the high (abstracted) level of hierarchical control.

In the following theorem, we show the relation between nonblockingness of a coordinated system and conditional decomposability of that system. First, however, we need the following auxiliary lemmas.

Lemma 10 (Proposition 4.1 in [16]). *Let $L \subseteq E^*$ be a language and $P_k : E^* \to E_k^*$ be a projection with $E_k \subseteq E$, for some alphabet E. Then, $P_k(\overline{L}) = \overline{P_k(L)}$.*

Lemma 11. *Let E be an alphabet, $L \subseteq E^*$ be a language, and $P_k : E^* \to E_k^*$ be a projection with $E_k \subseteq E$, for some alphabet E. Then, $L \parallel P_k(L) = L$.*

PROOF. By definition, $L \parallel P_k(L) = L \cap P_k^{-1} P_k(L)$, and it is not hard to see that $L \subseteq P_k^{-1} P_k(L)$. □

Theorem 12. *Let $n \geq 2$, and let G_i, for $i = 1, 2, \ldots, n$, be generators over the alphabets E_i, $i = 1, 2, \ldots, n$, respectively. Let G_k be a generator over an alphabet E_k such that $E_s \subseteq E_k \subseteq \bigcup_{i=1}^{n} E_i$. Then, the coordinated system $G = G_1 \parallel G_2 \parallel \ldots \parallel G_n \parallel G_k$ is nonblocking if and only if the following conditions both hold:*

1. *$G_i \parallel G_k \parallel \parallel_{j \neq i} P_k(G_j)$, for all $i = 1, 2, \ldots, n$, are nonblocking and*
2. *$\overline{L_m(G)}$ is conditionally decomposable with respect to the alphabets $E_1, E_2, \ldots, E_n, E_k$.*

PROOF. The following always holds for all $i = 1, 2, \ldots, n, n \geq 2$:

$$\begin{aligned}
\overline{L_m(G)} &\subseteq P_{1+k}(\overline{L_m(G)}) \parallel \ldots \parallel P_{n+k}(\overline{L_m(G)}) \\
&\subseteq P_{1+k}(L(G)) \parallel \ldots \parallel P_{n+k}(L(G)) \\
&= L(G_1 \parallel G_k \parallel P_k(G_2 \parallel G_3 \parallel \ldots \parallel G_n)) \\
&\quad \parallel L(G_2 \parallel G_k \parallel P_k(G_1 \parallel G_3 \parallel \ldots \parallel G_n)) \\
&\quad \vdots \\
&\quad \parallel L(G_n \parallel G_k \parallel P_k(G_1 \parallel G_2 \parallel \ldots \parallel G_{n-1})) \\
&= L(G) \,,
\end{aligned} \tag{3}$$

where the last equation follows from the idempotent property of the parallel composition and Lemma 11. If the language $\overline{L_m(G)}$ is nonblocking, then the inclusions become equalities. Thus, from the first equality, we get that the language $\overline{L_m(G)}$ is conditionally decomposable as required in item 2 of the theorem. Similarly, for all $i = 1, 2, \ldots, n$,

$$\begin{aligned}
P_{i+k}(\overline{L_m(G)}) = \overline{P_{i+k}(L_m(G))} &= \overline{L_m(G_i \parallel G_k) \parallel P_{i+k}(L_m(\parallel_{j \neq i} G_j))} \\
&= \overline{L_m(G_i \parallel G_k \parallel P_{i+k}(\parallel_{j \neq i} G_j)} \\
&\subseteq \overline{L_m(G_i) \parallel \overline{L_m(G_k)} \parallel \overline{P_{i+k}(\parallel_{j \neq i} L_m(G_j))}} \\
&\subseteq L(G_i) \parallel L(G_k) \parallel P_{i+k}(\parallel_{j \neq i} L(G_j)) \\
&= P_{i+k}(L(G)) \,,
\end{aligned}$$

where the first equality holds by Lemma 10, the second equality holds by Lemma 4 because we project to the alphabet $E_i \cup E_k$ that includes the intersection of $E_i \cup E_k$ and $\bigcup_{j \neq i} E_j$, namely E_k. Finally, the last equality holds by the same argument as the second equality. Hence, if the global plant is nonblocking, the inclusions become equalities, which means that the subsystems $G_i \parallel G_k \parallel P_{i+k}(\parallel_{j \neq i} G_j) = G_i \parallel G_k \parallel \parallel_{j \neq i} P_k(G_j)$ are nonblocking.

On the other hand, from the assumptions 1 and 2 we immediately get that both inclusions in (3) are equalities. Thus, the implication holds. $\qquad \square$

Note that Condition 2 of Theorem 12 does not hold in general because one inclusion of conditional decomposability, namely $\overline{L_m(G)} \subseteq P_{1+k}(\overline{L_m(G)}) \parallel P_{2+k}(\overline{L_m(G)})$, can be strict. Thus, the prefix closure of the marked language $\overline{L_m(G_1 \parallel G_2 \parallel G_k)}$ of the coordinated system consisting of subsystems G_1 and G_2 and a coordinator G_k is not in general conditionally decomposable with respect to alphabets E_1, E_2, E_k as demonstrated in the following example.

Example 13. *Consider two subsystems G_1 and G_2, and a coordinator G_k as depicted in Figure 4, where the corresponding alphabets are $E_1 = \{a, b, d\}$, $E_2 = \{a, c, d\}$, and $E_k = \{a, d\}$. Then, we can consider the string cacb*
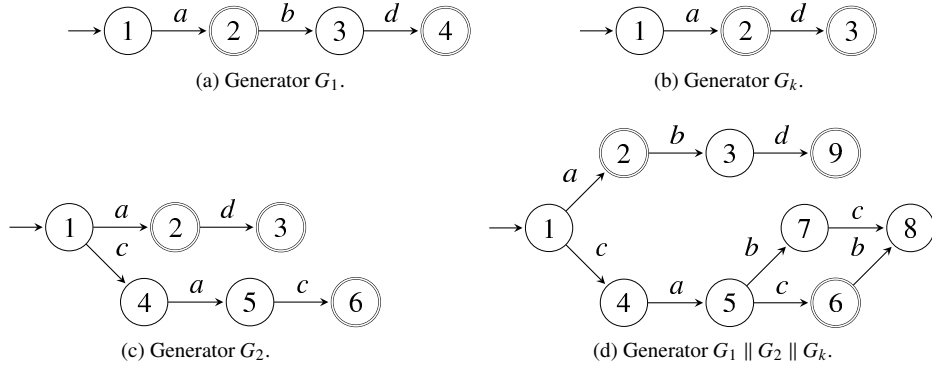

(a) Generator G_1.

(b) Generator G_k.

(c) Generator G_2.

(d) Generator $G_1 \parallel G_2 \parallel G_k$.

Figure 4: Generators G_1, G_2, G_k, and $G_1 \parallel G_2 \parallel G_k$.

and see that its projection $P_{1+k}(cacb) = ab$ belongs to the language $P_{1+k}(\overline{L_m(G_1 \parallel G_2 \parallel G_k)})$, and the projection $P_{2+k}(cacb) = cac$ belongs to the language $P_{2+k}(\overline{L_m(G_1 \parallel G_2 \parallel G_k)})$. However, this means that the string cacb belongs to the composition $P_{1+k}(\overline{L_m(G_1 \parallel G_2 \parallel G_k)}) \parallel P_{1+k}(\overline{L_m(G_1 \parallel G_2 \parallel G_k)})$. On the other hand, the string cacb is not a prefix of any string belonging to the marked language $\overline{L_m(G_1 \parallel G_2 \parallel G_k)}$ of the coordinated system as is easily seen in Figure 4(d). Thus, the language is not conditionally decomposable with respect to alphabets E_1, E_2, E_k. $\qquad \diamond$

Note that it follows from (3) that conditional decomposability is a weaker condition than nonblockingness. This is because conditional decomposability requires only the first inclusion to be equality, while nonblockingness requires both the inclusions to be equalities. The fundamental question is whether it is possible to decide in a distributed way without computing the whole plant whether $\overline{L_m(\parallel_{i=1}^n G_i \parallel G_k)}$ is conditionally decomposable. The algorithm described in the previous section requires the computation of the whole plant.

A specific choice of $L_m(G_k) \subseteq \bigcap_{i=1}^n P_k(L_m(G_i))$, respectively $L_m(G_k) = \bigcap_{i=1}^n P_k(L_m(G_i))$, yields Corollaries 14 and 15 below, respectively.

Corollary 14. *Let $G_1, G_2, \ldots, G_n, G_k$ be nonblocking generators over the alphabets $E_1, E_2, \ldots, E_n, E_k$, respectively, such that $E_s \subseteq E_k \subseteq \bigcup_{i=1}^n E_i$. Assume that $L_m(G_k) \subseteq \bigcap_{i=1}^n P_k(L_m(G_i))$. Then, the coordinated system $G = G_1 \parallel G_2 \parallel \ldots \parallel G_n \parallel G_k$ is nonblocking if and only if the following conditions both hold:*

1. *$G_i \parallel G_k$ are nonblocking, for all $i = 1, 2, \ldots, n$, and*
2. *$\overline{L_m(G)}$ is conditionally decomposable with respect to the alphabets $E_1, E_2, \ldots, E_n, E_k$.*

PROOF. By the assumption, $L_m(G_k) \subseteq \bigcap_i P_k(L_m(G_i))$. Applying the prefix closure to the previous inclusion results in the inclusion $L(G_k) = \overline{L_m(G_k)} \subseteq \overline{P_k(\bigcap_i L_m(G_i))} \subseteq \bigcap_i P_k(\overline{L_m(G_i)}) = \bigcap_i P_k(L(G_i)) = \parallel_i P_k(L(G_i))$. From this, it follows that $L(G_k) \parallel P_k(L(G_i)) = L(G_k)$, for $i = 1, 2, \ldots, n$, which implies that $G_i \parallel G_k \parallel \parallel_{j \neq i} P_k(G_j) = G_i \parallel G_k$. Thus, item 1 of Theorem 12 reduces to item 1 of this corollary. $\qquad \square$

10

Corollary 15. *Let $G_1, G_2, \ldots, G_n, G_k$ be nonblocking generators over the alphabets $E_1, E_2, \ldots, E_n, E_k$, respectively, such that $E_s \subseteq E_k \subseteq \bigcup_{i=1}^{n} E_i$, and assume that $L_m(G_k) = \bigcap_{i=1}^{n} P_k(L_m(G_i))$. Then, the coordinated system $G = \|_{i=1}^{n} G_i \| G_k$ is nonblocking if and only if the following conditions both hold:*

1. *$G_i \| G_k$ are nonblocking, for all $i = 1, 2, \ldots, n$, and*
2. *$\overline{L_m(G_1 \| G_2 \| \ldots \| G_n)}$ is conditionally decomposable with respect to alphabets $E_1, E_2, \ldots, E_n, E_k$.*

PROOF. The proof follows immediately from the previous corollary and the fact that $\|_i L_m(G_i) \| L_m(G_k) = \left(\|_i L_m(G_i) \right) \| \left(\|_i P_k(L_m(G_i)) \right)$, which is equal to $\|_i L_m(G_i)$ by Lemma 11, which reduces item 2 of Corollary 14 to the form of item 2 of this corollary. $\square$

The last corollary is particularly interesting because the coordinated modular discrete-event system coincides with the original plant and, therefore, nonblockingness of the original plant itself can be checked using the approach based on a coordinator, provided that we can verify item 2 in a distributed way.

The approach discussed above is based on projections, and the only known sufficient condition ensuring that the projected automaton is smaller with respect to the number of states than the original one is the observer property mentioned below. This topic requires further investigation because the observer property is only a sufficient condition, not necessary; there are examples of projected automata that are smaller than original automata without the projections satisfying the observer property. For completeness, however, we now discuss the case of projections satisfying the observer property and show that it corresponds to the known results discussed in [16] and in references therein.

Finally, we mention that in practice one central coordinator is particularly useful for loosely coupled subsystems, where the interaction between the subsystems (via synchronisation) is not too strong. Otherwise, a general multilevel hierarchy approach should be adopted, where the subsystems are aggregated into groups that are only loosely coupled. This is, however, very technical and left for a future study.

5.1. Observer property

The previous results are of interest in the case the projected systems $P_k(G_i)$, for $i = 1, 2, \ldots, n$, are significantly smaller than the original systems G_i. So far, the only known condition ensuring this is a so-called *observer property*.

Definition 16 (Observer property). *Let $E_k \subseteq E$ be alphabets. A projection $P_k : E^* \rightarrow E_k^*$ is an L-observer for a language $L \subseteq E^*$ if the following holds: for all strings $t \in P(L)$ and $s \in \overline{L}$, if $P(s)$ is a prefix of t, then there exists $u \in E^*$ such that $su \in L$ and $P(su) = t$.*

The following lemma proves that if the projections are observers, then item 2 of the previous results can be eliminated because it is always satisfied.

Lemma 17. *Let G_i, $1, 2, \ldots, n$, $n \geq 2$, and G_k be generators over the alphabets E_i, $i = 1, 2, \ldots, n$, and E_k, respectively, such that $E_s \subseteq E_k \subseteq \bigcup_i E_i$, and denote $G = \|_i G_i \| G_k$. If the projections P_k^{i+k} are $P_{i+k}(L_m(G))$-observers, for $i = 1, 2, \ldots, n$, then the language $\overline{L_m(G)}$ is conditionally decomposable with respect to E_i, $i = 1, 2, \ldots, n$, and E_k.*

PROOF. By Lemma 10, showing the first equality, it holds in general that

$$\overset{n}{\underset{i=1}{\|}} P_{i+k}(\overline{L_m(G)}) = \overset{n}{\underset{i=1}{\|}} \overline{P_{i+k}(L_m(G))} \supseteq \overline{\overset{n}{\underset{i=1}{\|}} P_{i+k}(L_m(G))}$$

$$= \overline{\overset{n}{\underset{i=1}{\|}} L_m\left(G_i \| G_k \| \underset{j \neq i}{\|} P_k(G_j) \right)} = \overline{L_m(G)} .$$

(4)

The last equality follows from the commutativity of the synchronous product and Lemma 11. By [24], it holds that $\|_{i=1}^{n} \overline{P_{i+k}(L_m(G))} = \overline{\|_{i=1}^{n} P_{i+k}(L_m(G))}$ if and only if $\|_{i=1}^{n} \overline{P_k(L_m(G))} = \overline{\|_{i=1}^{n} P_k(L_m(G))}$, and the later equality is obviously satisfied. Thus, the former equality implies by (4) that the language $\overline{L_m(G)}$ is conditionally decomposable with respect to alphabets E_1, E_2, E_k, which was to be shown. $\square$

As mentioned in the previous proof, when we consider all the assumptions, Feng [16] (see also the references therein) has shown that if the projection P_k is an observer for L_1 and L_2, then $L_1 \parallel L_2$ is nonconflicting if and only if $P_k(L_1) \parallel P_k(L_2)$ is nonconflicting. This is generalized to arbitrary components in [24]. Note that using this property on item 1 of Corollary 15, together with the previous lemma and the fact that the observers preserve parallel composition, [24], results in the following corollary, which generalizes the results shown in [16] for two components.

Corollary 18. *Let G_i, $1, 2, \ldots, n$, $n \geq 2$, and G_k be nonblocking generators over the alphabets E_i, $i = 1, 2, \ldots, n$, and E_k, respectively, such that $E_s \subseteq E_k \subseteq \bigcup_i E_i$, and assume that $L_m(G_k) = \bigcap_i P_k(L_m(G_i))$ and $L(G_k) = \bigcap_i P_k(L(G_i))$. Assume that the projections P_k^i are $L_m(G_i)$-observers, for $i = 1, 2, \ldots, n$. Then, the coordinated system $\parallel_i G_i \parallel G_k$ is nonblocking if and only if G_k is nonblocking.*

This works because the projection is an observer. However, there are languages which are conditionally decomposable, but the projections from Lemma 17 are not observers. For instance, consider a language $L = \{ba, cdb, dcb\}$. It can be verified that L is conditionally decomposable with respect to the alphabets $E_1 = \{a, b, c\}$, $E_2 = \{a, b, d\}$, and $E_k = \{a, b\}$, and that the projections P_k^{i+k} are not $P_{i+k}(L)$-observers, for $i = 1, 2$. Note that $P_{1+k}(L) = \{ba, cb\}$ and $P_{2+k}(L) = \{ba, db\}$. Then, for $t = b$ and $s = cb$ (for $i = 1$, or $s = db$ for $i = 2$), there is no extension of cb such that $P_k^{1+k}(cb) = ba$. Hence, the projections are not observers. For that reason, we consider in this paper a more general assumption that the projections are such that the projected generators are smaller than the original generators. Note that the conditions under which this is true still need to be investigated. Finally, note that for the verification whether the subsystems $G_i \parallel G_k \parallel \parallel_{j \neq i} P_k(G_j)$ are nonblocking, the methods presented in [25, 26] can be used, combined with further usage of Binary Decision Diagrams [27] or state-tree structures [28] to perform the calculations.

6. Conclusion

The main contributions of this paper are polynomial-time algorithms for the verification whether a language is conditionally decomposable and for an extension of the coordinator alphabet E_k. Our approach to extend the alphabet E_k is based on the successive addition of events to the alphabet E_k. Another approach has recently been discussed in [29], where the problematic transitions are identified, and the events of these transitions are renamed. From the viewpoint of applications, however, our approach can directly be used in coordination control for which it has primarily been developed. On the other hand, the approach from [29] has so far no direct applications in the coordination control framework, which is under investigation. Nevertheless, the algorithms presented here can also be used for the approach presented in [29].

Particularly valuable is the property that algorithms for checking conditional decomposability of a language with respect to alphabets is linear in the number of alphabets (that corresponds to local controllers in coordination control). No such results are known for co-observability (the notion playing a central role in decentralized control) and the related property of decomposability. It is well-known that co-observability is equivalent to decomposability under some reasonable assumptions on locally controllable and locally observable alphabets. Since conditional decomposability can be seen as decomposability with respect to particular alphabets (enriched by the coordinator events), it appears that our results about conditional decomposability will have impact on decentralized control with communicating supervisors. Indeed, co-observability is ensured by a special types of communication (which corresponds to enriching the sets of locally observable events such that a specification language becomes co-observable) in a similar way as decomposability is imposed by enriching the alphabets of local supervisors.

The paper also compares the property of conditional decomposability to nonblockingness of a coordinated system. The current low complexity tests of practical interest are based on the observer property because it is the only known condition ensuring that the projected generator is smaller than the original one. However, this is only a sufficient condition and further investigation is needed. It is our plan to further investigate the construction procedures for designing coordinators for nonblockingness that are as small as possible and we will combine these results with those obtained in coordination control for safety so that both nonblockingness and safety issues can be efficiently handled using coordination control.

Acknowledgments

References

[1] P. J. Ramadge, W. M. Wonham, Supervisory control of a class of discrete event processes, SIAM J. Control Optim. 25 (1987) 206–230.

[2] Y. Willner, M. Heymann, Supervisory control of concurrent discrete-event systems, Internat. J. Control 54 (1991) 1143–1169.

[3] K. Rudie, W. Wonham, Supervisory control of communicating processes, in: Proc. of International Symposium on Protocol Specification, Testing and Verification X, Elsevier Science Publishers, Amsterdam, The Netherlands, 1990, pp. 243–257.

[4] K. Rudie, W. M. Wonham, Think globally, act locally: Decentralized supervisory control, IEEE Trans. Automat. Control 37 (1992) 1692–1708.

[5] S. Jiang, R. Kumar, Decentralized control of discrete event systems with specializations to local control and concurrent systems, IEEE Trans. Syst. Man Cybern. B 30 (2000) 653–660.

[6] M. Karimadini, H. Lin, Decomposability of global tasks for multi-agent systems, in: Proc. of CDC 2010, pp. 4192–4197.

[7] B. Gaudin, H. marchand, An efficient modular method for the control of concurrent discrete event systems: A language-based approach, Discrete Event Dyn. Syst. 17 (2007) 179–209.

[8] J. Komenda, J. H. van Schuppen, B. Gaudin, H. Marchand, Supervisory control of modular systems with global specification languages, Automatica 44 (2008) 1127–1134.

[9] J. Komenda, J. H. van Schuppen, Coordination control of discrete event systems, in: Proc. of WODES 2008, pp. 9–15.

[10] J. Komenda, T. Masopust, J. H. van Schuppen, Synthesis of controllable and normal sublanguages for discrete-event systems using a coordinator, Systems Control Lett. 60 (2011) 492–502.

[11] J. Komenda, T. Masopust, J. H. van Schuppen, Supervisory control synthesis of discrete-event systems using a coordination scheme, Automatica 48 (2012) 247–254.

[12] G. Jirásková, T. Masopust, State complexity of projected languages, in: Proc. of DCFS 2011, volume 6808 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 198–211.

[13] C. G. Cassandras, S. Lafortune, Introduction to discrete event systems, Springer, 2nd edition, 2008.

[14] A. Salomaa, Formal languages, Academic Press, New York, 1973.

[15] W. M. Wonham, Supervisory control of discrete-event systems, 2011. Lecture notes, Department of electrical and computer engineering, University of Toronto. [Online]. Available: http://www.control.utoronto.ca/DES/.

[16] L. Feng, Computationally Efficient Supervisor Design for Discrete-Event Systems, Ph.D. thesis, University of Toronto, 2007.

[17] L. Feng, W. M. Wonham, Computationally efficient supervisor design: Abstraction and modularity, in: Proc. of WODES 2006, Ann Arbor, USA, pp. 3–8.

[18] J. Komenda, T. Masopust, J. H. van Schuppen, Synthesis of safe sublanguages satisfying global specification using coordination scheme for discrete-event systems, in: Proc. of WODES 2010, pp. 436–441.

[19] M. Sipser, Introduction to the theory of computation, PWS Publishing Company, Boston, 1997.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, The MIT Press, 3rd edition, 2009.

[21] G. Karakostas, R. J. Lipton, A. Viglas, On the complexity of intersecting finite state automata and NL versus NP, Theoret. Comput. Sci. 302 (2003) 257–274.

[22] K. Wong, On the complexity of projections of discrete-event systems, in: Proc. of WODES 1998, Cagliari, Italy, pp. 201–206.

[23] T. Moor, et al., libFAUDES – discrete event systems library, February 2012. [Online].
Available: http://www.rt.eei.uni-erlangen.de/FGdes/faudes/index.html.

[24] P. N. Pena, J. E. R. Cury, S. Lafortune, Verification of nonconflict of supervisors using abstractions, IEEE Trans. Automat. Control 54 (2009) 2803–2815.

[25] H. Flordal, R. Malik, Modular nonblocking verification using conflict equivalence, in: Proc. of WODES 2006, pp. 100–106.

[26] H. Flordal, R. Malik, Compositional verification in supervisory control, SIAM J. Control Optim. 48 (2009) 1914–1938.

[27] R. E. Bryant, Symbolic boolean manipulation with ordered binary-decision diagrams, ACM Comput. Surv. 24 (1992) 293–318.

[28] C. Ma, W. M. Wonham, Nonblocking Supervisory Control of State Tree Structures, Lecture Notes in Control and Information Sciences, Springer, 2005.

[29] T. Masopust, S. L. Ricker, Another approach to conditional decomposability for discrete-event systems, 2012. Manuscript.