Systems & Control Letters 61 (2012) 1260–1268

Contents lists available at SciVerse ScienceDirect

Systems & Control Letters

journal homepage: www.elsevier.com/locate/sysconle

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

ARTICLE INFO

Article history:
Received 8 December 2011
Received in revised form
20 March 2012
Accepted 24 July 2012
Available online 12 November 2012

Keywords:
Discrete-event system
Coordination control
Conditional decomposability

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 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 in general settings is also discussed in this paper. It is shown that conditional decomposability is a weaker condition than nonblockingness.

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]; it 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 $i = 1, 2, \ldots, n$, $P_i : \left(\bigcup_{i=1}^{n} E_i\right)^* \to E_i^*$ 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 [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 $K = \|_{i=1}^{n} P_i(K) \parallel L$: separability is then decomposability where $L = \left(\bigcup_{i=1}^{n} E_i\right)^*$ 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 [6].

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] is 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

* Corresponding author. Tel.: +420 222090784; fax: +420 541218657.
E-mail addresses: komenda@ipm.cz (J. Komenda), masopust@math.cas.cz (T. Masopust), J.H.van.Schuppen@cwi.nl (J.H. van Schuppen).

0167-6911/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysconle.2012.07.013

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 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 co-observability) suffer from 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 $\mathtt{trim}(G)$ denote the *trim* of G, that is, a generator $\mathtt{trim}(G)$ such that $\overline{L_m(\mathtt{trim}(G))} = L(\mathtt{trim}(G)) = \overline{L_m(G)}$. 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 $\mathtt{trim}(G)$; see [13,15]. A generator G is said to be *nonblocking* if $\overline{L_m(G)} = L(G)$. Thus, $\mathtt{trim}(G)$ 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 $P_i : \left(\bigcup_j E_j\right)^* \to E_i^*$.

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. [15–17].

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 [9–11,18] for applications and further discussion concerning this concept.

Definition 1 (*Conditional Decomposability*)**.** A language K over an alphabet $E_1 \cup E_2 \cup \cdots \cup E_n$, where $n \geq 2$, is said to be *conditionally decomposable with respect to* $E_1, E_2, \ldots, E_n$, *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

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

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.