Another Approach to Conditional Decomposability for Discrete-Event Systems

Tomáš Masopust and Laurie Ricker

***Abstract*— The notion of decomposability with respect to alphabets E_1 and E_2 is well-known and important in modular control of discrete-event systems. In coordination control, a weakened version of decomposability, so-called conditional decomposability, was introduced. A language is conditionally decomposable with respect to alphabets E_1, E_2, E_k if it is decomposable with respect to $E_1 \cup E_k$ and $E_2 \cup E_k$. There always exists such an alphabet E_k for which the language is conditionally decomposable. However, when looking for the alphabet E_k, the problematic events are identified and added to E_k, but this means that instead of only the problematic transitions we make all other transitions with the same label observable. In this paper, we further weaken the condition so that we identify the problematic transitions and define a projection from transitions rather than from the events.**

I. INTRODUCTION

Decomposability of a specification language K with respect to alphabets E_1 and E_2 and the operation of parallel composition is a well-known and very important property in modular control of discrete-event systems. It was introduced in [11], [10] and further studied in, e.g., [3]. Recently, it has also been extended to automata as an automaton decomposability in, e.g., [4]. Unfortunately, it is not always the case that K is decomposable with respect to the given alphabets. Recently, a weakened version of decomposability, so-called *conditional decomposability*, has been introduced in [9] and studied in [6], [8] in the context of coordination supervisory control.

A language is conditionally decomposable with respect to alphabets E_1, E_2, and E_k if it is decomposable with respect to $E_1 \cup E_k$ and $E_2 \cup E_k$. It is not hard to see that there always exists such an alphabet E_k for which the language is conditionally decomposable. Of course, we could take $E_k = E_1 \cup E_2$, but this set should be the smallest one, or at least reasonably small. Note that there exists a polynomial-time algorithm for finding some alphabet E_k for which the language is conditionally decomposable, see [7]; however, the resulting alphabet is not always the minimal one. The complexity of the computation of the minimal set E_k for which the language is conditionally decomposable is an open problem.

In the approach presented in [7], the computation of an alphabet E_k, necessitates that when events that give rise to the violation of conditional decomposability are identified, the event is added to E_k. Thus, even though there may be some occurrences of the event that do not affect whether or not the language is conditionally decomposable, at an event level, all occurrences of the event are made observable by adding the event to E_k.

In this paper, we discuss this problem and suggest a method to characterize the problematic transitions and add only these occurrences of an event to E_k to resolve violations of conditional decomposability. To consider only the relevant transitions, we suggest the use of a homomorphism based on the transitions rather than on the events. An equivalent approach is to first identify the problematic transitions, rename the transition labels with a unique identifier, using a mask so that these new events are shared by both subsystems, and then add the renamed events to the alphabet E_k. To facilitate this, we use homomorphisms instead of projections.

T. Masopust is with the Institute of Mathematics, Academy of Sciences of the Czech Republic, Žižkova 22, 616 62 Brno, Czech Republic. Email: masopust@math.cas.cz

L. Ricker is with the Department of Mathematics & Computer Science, Mount Allison University, Sackville, NB, Canada. Email: lricker@mta.ca

II. PRELIMINARIES AND DEFINITIONS

In this paper, we assume that the reader is familiar with the basic notions and concepts of supervisory control theory, see [2], and with automata theory, see [12]. For an alphabet (finite nonempty set) E, E^* denotes the free monoid generated by E, where the unit of E^*, the empty string, is denoted by ε.

A *natural projection* $P : E^* \to E_o^*$, where $E_o \subseteq E$ are alphabets, is a homomorphism defined so that $P(a) = \varepsilon$, for $a \in E \setminus E_o$, and $P(a) = a$, for $a \in E_o$. In other words, homomorphism is a concatenative mapping, that is, $P(\varepsilon) = \varepsilon$, and for any event $a \in E$ and any string $w \in E^*$, $P(aw) = P(a)P(w)$. The inverse projection $P^{-1} : E_o^* \to 2^{E^*}$ is then naturally defined as $P^{-1}(s) = \{t \in E^* \mid P(t) = s\}$.

In what follows, we use the notation P^i_j to denote the projection from E_i to E_j, that is, $P^i_j : E_i^* \to E_j^*$. In addition, we use the notation $E_{i+j} = E_i \cup E_j$, and, thus, P^{i+j}_k denotes the projection from E_{i+j} to E_k. If $E = E_1 \cup E_2$, then P_j denotes the projection from E^* to E_j^*.

A *generator* is a quintuple $G = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is an *input alphabet*, $\delta : Q \times \Sigma \to Q$ 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, the transition function δ is extended to a function from $Q \times \Sigma^*$ to Q. The language *generated* by G is defined as the set $L(G) = \{w \in \Sigma^* \mid \delta(q_0, w) \in Q\}$, and the language *marked* by G is defined as the set $L_m(G) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. 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 $a \in \Sigma$.

A *(regular) language* over an alphabet E is a subset of E^* such that there exists a generator G with $L_m(G) = L$. 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 $L = \overline{L}$.

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

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

A similar definition in terms of generators follows. Let $G_1 = (X_1, E_1, \delta_1, x_{01}, X_{m1})$ and $G_2 = (X_2, E_2, \delta_2, x_{02}, X_{m2})$ 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 $(X_1 \times X_2, E_1 \cup E_2, \delta, (x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$, where

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

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, e.g., [2].

Let G be a generator and P be a projection, then the projected generator $P(G)$ is defined as 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 [2] or [13].

A. Conditional decomposability

Now, we recall the main concept of interest of this paper, the concept of conditional decomposability. See also [5], [8], [6], [9] for the applications and further discussion concerning this concept.

Definition 1 (Conditional decomposability): A language $K \subseteq (E_1 \cup E_2)^*$ is *conditionally decomposable with respect to alphabets* E_1, E_2, E_k, where $E_1 \cap E_2 \subseteq E_k \subseteq E_1 \cup E_2$, if

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

It can easily be shown that if the language K is given as a parallel composition of two languages (over the required alphabets), then it is conditionally decomposable. Thus, if K is conditionally decomposable, the pair of languages $P_{1+k}(K)$ and $P_{2+k}(K)$ is the smallest decomposition of K with respect to the corresponding alphabets.

On the other hand, when a language K is not conditionally decomposable, it is always possible to update E_k with events from E such that K is subsequently conditionally decomposable, cf. [7]. Here we take a different approach to making the language conditionally decomposable. When K is not conditionally decomposable with respect to E_1, E_2 and E_k, we consider the addition of specific occurrences of events in δ (i.e., transitions that violate the property of conditional decomposability and their observationally equivalent counterparts) to the set of transition labels made observable through E_k.

III. Transition-based Conditional Decomposability

In this section, we present a structure on which we can identify the transitions that give rise to violations of conditional decomposability. First, recall from [7] the result from which we will build our new approach.

Theorem 2: Let G be a generator. Language $L_m(G)$ is conditionally decomposable with respect to alphabets E_1, E_2, and E_k if and only if $L_m(\tilde{G}) \subseteq \tilde{P}^{-1}(L_m(G))$, where $\tilde{G} = f_{1+k}(G) \parallel f_{2+k}(G)$, for some homomorphisms f_{i+k}, $i = 1,2$, and a projection $\tilde{P} : (E \cup \tilde{E}_{1+k} \cup \tilde{E}_{2+k})^* \to E^*$.

A. An auxiliary structure $\mathscr{U}$

The main idea of our approach is that instead of adding all occurrences of an event to E_k, we want to select only a subset of the occurrences of events that lead to violations of conditional decomposability.

To do this, consider a language L over an alphabet E, marked by a generator G. We assume that we are given alphabets E_1, E_2, and E_k such that $E_1 \cap E_2 \subseteq E_k \subseteq E_1 \cup E_2$, where $E = E_1 \cup E_2$.

It will be useful to augment each alphabet with the addition of the empty string ε, where such an addition is denoted by a superscript of ε (e.g., $E^\varepsilon = E \cup \{\varepsilon\}$).

To detect violations of conditional decomposability, we first update the transition function δ in G so that there are self-loops of ε at each state, thus requiring the alphabet to be E^ε. Next, we construct two copies of the generator $G^\varepsilon = (Q, E^\varepsilon, \delta, q_0, F)$, denoted G^ε_{1+k} and G^ε_{2+k}, with alphabets E^ε_{1+k} and E^ε_{2+k}, respectively, from which we build a product

$$\mathscr{U} = G^\varepsilon \times G^\varepsilon_{1+k} \times G^\varepsilon_{2+k} = (\mathbb{X}, \mathbb{E}, \Delta, x_0, \mathbb{F}, \mathbb{X}_{conD})\,,$$

where $\mathbb{X} \subseteq (Q \cup \{\Box\})^3$ is a finite set of states and the symbol $\Box$ represents a destination state for an undefined transition; $\mathbb{E} \subseteq E^\varepsilon \times E^\varepsilon \times E^\varepsilon$ is a finite alphabet of synchronization vectors (defined below); Δ is the transition relation (also defined below); $x_0 = (q_0, q_0, q_0)$ is the initial state; $\mathbb{F} \subseteq Q \times F \times F$ is a set of final states; and $\mathbb{X}_{conD} = \{x \in \mathbb{X} \mid x(0) = \Box\}$ is a set of illegal configurations of states that encode violations of conditional decomposability.

B. Alphabet of $\mathscr{U}$

The alphabet $\mathbb{E} \subseteq E^\varepsilon \times E^\varepsilon \times E^\varepsilon$ of the structure $\mathscr{U}$ is a set of synchronization vectors, cf. [1]. Synchronization vectors are based on the synchronization of an event in E and the same event in each of the local alphabets E_{i+k}, for $i = 1,2$. Each vector is represented by a triple of events

$$v = \langle v(0), v(1), v(2) \rangle\,.$$

There are two different categories of vectors to construct, corresponding to (a) events in E but not in E_{i+k}; and (b) events in E_{1+k} and/or E_{2+k}.

For all $e \in E \setminus E_{i+k}$ (for $i = 1,2$) and for $j = 0,1,2$:

$$v(j) = \begin{cases} e\,, & \text{if } j = i, \\ \varepsilon\,, & \text{otherwise}. \end{cases}$$

Similarly, for all $e \in E_{i+k}$ (for $i = 1,2$) and for $j = 0,1,2$:

$$v(j) = \begin{cases} e, & \text{if } e \in E_{j+k} (j \neq 0) \text{ or } j = 0, \\ \varepsilon, & \text{otherwise}. \end{cases}$$

C. Transition function of $\mathscr{U}$

The transition relation $\Delta : \mathbb{X} \times \mathbb{E} \to \mathbb{X}$ is defined for all $v \in \mathbb{E}$ as follows:

$$\Delta(x,v) = \begin{cases} \big(\delta(x(0),v(0)), \delta(x(1),v(1)), \delta(x(2),v(2))\big), \\ \quad \text{if } \delta(x(0),v(0))! \\ \quad \wedge \delta(x(1),v(1))! \\ \quad \wedge \delta(x(2),v(2))!; \\ \big(\square, \delta(x(1),v(1)), \delta(x(2),v(2))\big), \\ \quad \text{if } \neg\delta(x(0),v(0))! \\ \quad \wedge \delta(x(1),v(1))! \\ \quad \wedge \delta(x(2),v(2))!; \\ \text{undefined}, \quad \text{otherwise}. \end{cases}$$

Moreover, note that the marked states of this structure are only those of the form $Q \times F \times F$. Using these marked states, we employ the standard *trim* procedure to remove all states from which no marked state is reachable.

D. Informal explanation

For $w \in \mathbb{E}^*$, where $w = v_0 v_1 \dots v_r$, we will find it convenient to denote the sequence generated by component j in $\mathscr{U}$ by $w(j) = v_0(j) \dots v_r(j)$, for $j = 0,1,2$.

What we need to check here is that we have $L_m(G) = P_{1+k}(L_m(G)) \parallel P_{2+k}(L_m(G))$. However, it is always true that $L_m(G) \subseteq P_{1+k}(L_m(G)) \parallel P_{2+k}(L_m(G))$. Therefore, we need to verify the opposite inclusion, namely

$$L_m(G) \supseteq P_{1+k}(L_m(G)) \parallel P_{2+k}(L_m(G)).$$

Intuitively, we use the structure $\mathscr{U}$ to simulate the computation of all three subsystems at the same time and to verify that whenever the components G_{1+k} and G_{2+k}, corresponding to the right-hand side of the inclusion, can make a step, this step must also be possible in G.

A violation of conditional decomposability is then detected in $\mathscr{U}$ when $\mathbb{X}_{conD} \neq \emptyset$, that is, there are accessible states in $x \in \mathbb{X}$ such that $x(0) = \square$. This corresponds to a situation when there are no more strings that can be generated by G, but at least one of G_{i+k} (for $i = 1,2$) continues to generate sequences based on what it considers to be the current state of G. Furthermore, these sequences are not part of the marked language of G.

Transitions that lead to a violation of conditional decomposability can be identified as follows. Let

$$\delta(x,\ell) = x'$$

be an incoming transition to a state in $x' \in \mathbb{X}_{conD} \neq \emptyset$. We add the occurrence of the transition

$$(x(j), \ell(j), x'(j))$$

where $\ell(j) \neq \varepsilon$ (for $j \in \{1,2\}$) to the set of problematic transitions of G_{i+k}, where $i = \{1,2\} \setminus \{j\}$, and relabel the event $\ell(j)$ with a new event

$$\widetilde{\ell(j)},$$

add this event to the alphabet $\tilde{E}_{i+k} = E_{i+k} \cup \{\widetilde{\ell(j)}\}$, and, further, define $P_i(\widetilde{\ell(j)}) = \ell(j)$.

IV. Example

We now illustrate our approach via an example.

Example 3: Consider the language $L_m(G)$ generated by G depicted in Fig. 1, where $E_1 = \{a,b,d\}$, $E_2 = \{a,c,d\}$, and $E_k = \{a,d\}$. Then the automata G_{1+k} and G_{2+k} are as in Fig. 2 and 3.

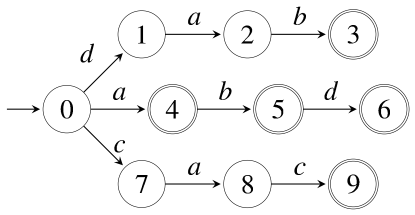

Fig. 1. Generator G.

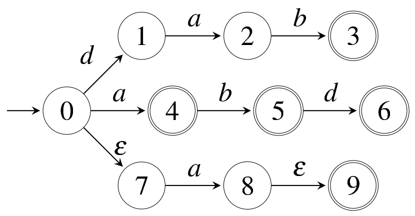

Fig. 2. Generator G_{1+k}.

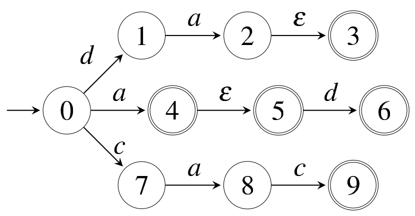

Fig. 3. Generator G_{2+k}.

Based on alphabets E_1, E_2 and E_k, we can construct the alphabet of synchronization vectors:

$$\mathbb{E} = \{\langle a,a,a\rangle, \langle b,b,\varepsilon\rangle, \langle \varepsilon,\varepsilon,b\rangle, \langle \varepsilon,c,\varepsilon\rangle, \langle c,\varepsilon,c\rangle, \langle d,d,d\rangle\}.$$

and the structure $\mathscr{U}$ for this example is shown in Fig. 4.

We have $\mathbb{X}_{conD} = \{(\square,5,8), (\square,5,9)\}$. To see why these states encode violations of conditional decomposability, consider the sequence that leads to state $x = (\square,5,9)$, namely

$$w = \langle c,\varepsilon,c\rangle \langle a,a,a\rangle \langle c,\varepsilon,c\rangle \langle b,b,\varepsilon\rangle.$$

Note that while both $w(1) = ab$ and $w(2) = cac$ are possible in $L_m(G)$, it is the case that the interleaving of the contributions of E^{ε}_{1+k} and E^{ε}_{2+k} produce $w(0) = cacb$ that is not in

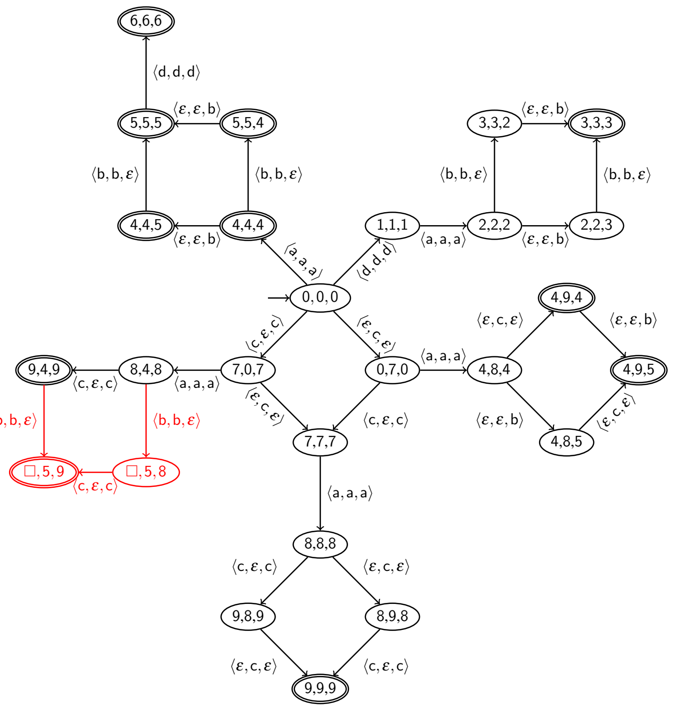

Fig. 4. $\mathscr{U}$ for Example 3, where violations of conditional decomposability are indicated in red.

$L_m(G)$. We know that this string is not possible in G because $x(0) = \square$.

To identify the transition(s) responsible for a violation, we examine incoming transitions to states in $\mathbb{X}_{conD}$. For illegal configuration $(\square, 5, 9)$, the incoming transition from state $(9, 4, 9)$ with label $\langle b, b, \varepsilon\rangle$ leads to a violation of conditional decomposability. In G_{1+k}, a transition from state 4 with b is still possible, and since G_{2+k} cannot observe occurrences of b, it cannot determine that b should not be allowed to occur at this point. If this occurrence of b was added to the alphabet of G_{2+k}, then the transition of $\langle b, b, \varepsilon\rangle$ from $(9, 4, 9)$ would never be defined. (By symmetry, we can determine the same transition is responsible for giving rise to state $(\square, 5, 8)$.)

We want to add the occurrence of b via transition $(4, b, 5)$ and any observationally-equivalent occurrences of b to E_{2+k}. To facilitate this, we relabel $(4, b, 5)$ as $(4, \tilde{b}, 5)$ in G, add $\tilde{b}$ to E_{2+k} with the understanding that $P_{i+k}(\tilde{b}) = b$, for $i \in \{1, 2\}$, and $P_k(\tilde{b}) = b$, see Fig. 5.

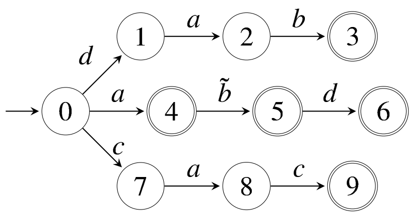

Fig. 5. Generator G with the relabelled problematic transition $(4, b, 5)$.

We now define the projection P from transitions to the alphabet $E_k \cup \{\tilde{b}\}$ as follows, see Fig. 6 and Fig. 7.

Then we can see that $L_m(P_{1+k}(G) \parallel P_{2+k}(G)) = L_m(G)$ as illustrated in Fig. 8 because the set $\mathbb{X}_{conD} = \emptyset$. ◇

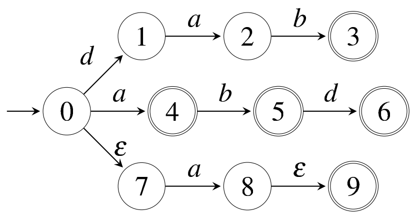

Fig. 6. Generator G_{1+k}.

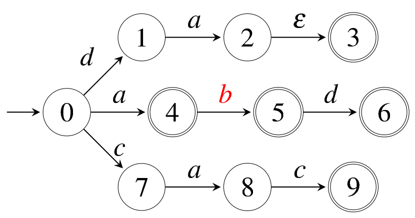

Fig. 7. Generator G_{2+k}.

Now we formally prove the correctness of our approach.

Theorem 4: $\mathbb{X}_{conD} = \emptyset$ if and only if $L_m(G)$ is conditionally decomposable with respect to alphabets E_1, E_2, E_k.

Proof: ($\Rightarrow$) For the sake of contradiction, suppose that the set $\mathbb{X}_{conD} = \emptyset$ and that the language $L_m(G)$ is not conditionally decomposable. Then there exist two shortest strings $t_1 \in L(P_{1+k}(G))$ and $t_2 \in L(P_{2+k}(G))$ such that the interleaving of t_1 and t_2 results in a prefix t of some sequence in $L_m(P_{1+k}(G) \| P_{2+k}(G)) \setminus L_m(G)$. By construction of the alphabet $\mathbb{E}$, there exists a string $w = v_1 v_2 \ldots v_{|w|} \in \mathbb{E}^*$ such that $w(0) = t$, $w(1) = t_1$, and $w(2) = t_2$. Let $\delta(q_0, t_1) = q_1$

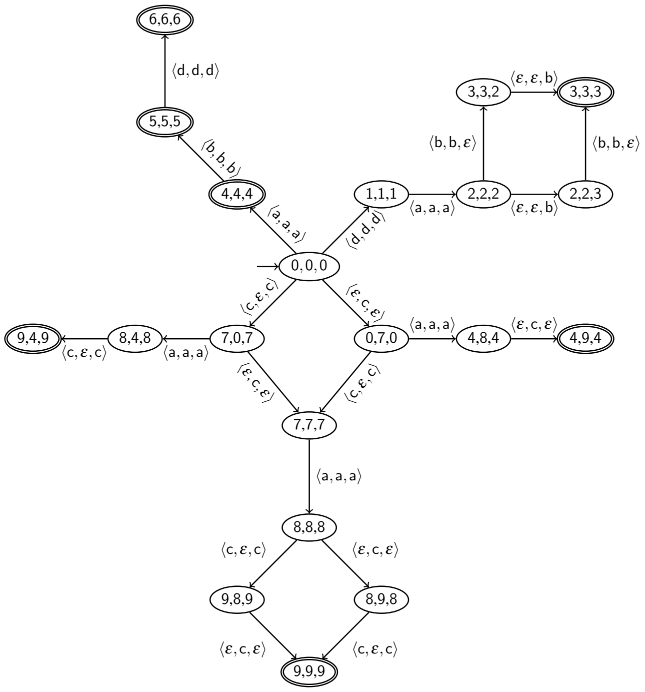

Fig. 8. $\mathscr{U}$ built with updated alphabets.

and $\delta(q_0,t_2)=q_2$. Then, in $\mathscr{U}$, $\Delta(x_0,w)=(\square,q_1,q_2)$. Thus we have $x\in\mathbb{X}_{conD}$, which is a contradiction.

($\Leftarrow$) On the other hand, suppose that the language $L_m(G)$ is conditionally decomposable and that the set $\mathbb{X}_{conD}\neq\emptyset$. Let $x\in\mathbb{X}_{conD}$. Let $w=v_1v_2\dots v_{|w|}\in\mathbb{E}^*$ such that $\Delta(x_0,w)=x$. Let $t=w(0)$ and $t_i=w(i)$, for $i=1,2$. Note that since $x(0)=\square$, we have $\neg\delta(q_0,w(0))!$. The only vectors on which t and t_i synchronize are those derived from events in $E\cap E_{i+k}$ and therefore $P_{i+k}(t)=P_{i+k}(t_i)=t_i$, for $i\in\{1,2\}$. Because $L_m(G)$ is conditionally decomposable, $t\in P_{1+k}^{-1}[P_{1+k}(t)]\cap P_{2+k}^{-1}[P_{2+k}(t)]\cap L_m(G)$ and it follows that $\delta(q_0,t)!$, a contradiction. ■

V. CONCLUSIONS AND FUTURE WORK

We have proposed a new approach to conditional decomposability, where we can construct a potentially smaller observable alphabet E_k (in terms of the cardinality of the set of observable transition labels) than was presented in [7]. The computational complexity of both the event-based approach and the transition-based approach is polynomial when we have $n=2$ subalphabets and exponential when these concepts are extended for $n>2$. The finite structure $\mathscr{U}$, which can be constructed in $O(|Q|^{n+1}|\Sigma^{\varepsilon}|^{n+1})$, is easily extended to handle $n>2$ subalphabets; however, we would like to avoid the explicit calculation of this structure, which requires additional study on the efficient implementation of these types of algorithmic techniques.

Subsequent work also includes a study of how to incorporate our approach into coordination control and other aspects of supervisory control of discrete-event systems that rely on the concept of decomposability.

ACKNOWLEDGMENTS

The research by T. Masopust was supported by the GAČR grant no. P202/11/P028, and by RVO: 67985840. L. Ricker was partially supported by NSERC. The authors extend their thanks to P. Darondeau for motivating and useful discussions.

REFERENCES

[1] A. Arnold, *Finite transition systems*. Prentice–Hall, 1994.

[2] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*, 2nd ed. Springer, 2008.

[3] S. Jiang and R. Kumar, "Decentralized control of discrete event systems with specializations to local control and concurrent systems," *IEEE Trans Syst Man Cybern B*, vol. 30, no. 5, pp. 653–660, 2000.

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

[5] J. Komenda, T. Masopust, and 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. [Online]. Available: http://www.ifac-papersonline.net/

[6] ——, "Synthesis of controllable and normal sublanguages for discrete-event systems using a coordinator," *Systems Control Lett.*, vol. 60, no. 7, pp. 492–502, 2011.

[7] ——, "On conditional decomposability," *CoRR*, vol. abs/1201.1733, 2012. [Online]. Available: http://arxiv.org/abs/1201.1733

[8] ——, "Supervisory control synthesis of discrete-event systems using a coordination scheme," *Automatica*, vol. 48, no. 2, pp. 247–254, 2012.

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

[10] K. Rudie and W. M. Wonham, "Think globally, act locally: Decentralized supervisory control," *IEEE Trans. Autom. Control*, vol. 37, no. 11, pp. 1692–1708, 1992.

[11] K. Rudie and W. Wonham, "Supervisory control of communicating processes," in *Protocol Specification, Testing and Verification, X*, H. Ural, P. R. L., and L. Logrippo, Eds. Elsevier Science Publishers B. V., 1990, pp. 243–257.

[12] A. Salomaa, *Formal languages*. New York: Academic Press, 1973.

[13] W. M. Wonham, "Supervisory control of discrete-event systems," http://www.control.utoronto.ca/cgi-bin/dldes.cgi, July 2011, Lecture notes, Department of Electrical and Computer Engineering, University of Toronto.