View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Chalmers Publication Library

Reduced-Order Synthesis of Operation Sequences

Mohammad Reza Shoaei, Sajed Miremadi, Kristofer Bengtsson and Bengt Lennartson
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
shoaei@chalmers.se

***Abstract*—In flexible manufacturing systems a large number of operations need to be coordinated and supervised to avoid blocking and deadlock situations. The synthesis of such supervisors soon becomes unmanageable for industrial manufacturing systems, due to state space explosion. In this paper we therefore develop some reduction principles for a recently presented model based on self-contained operations and sequences of operations. First sequential operation behaviors are identified and related operation models are simplified into one model. Then local transitions without interaction with other operation models are removed. This reduction principle is applied to a synthesis of non-blocking operation sequences, where collisions among moving devices are guaranteed to be avoided by a flexible booking process. The number of states in the synthesis procedure and the computation time is reduced dramatically by the suggested reduction principle.**

I. Introduction

Global competition is a driving force for manufacturing industry to rethink their strategies and methods. While continuously introducing new products is necessary to maintain and gain a market presence, reducing cost and time for producing the products remains a key challenge. A key issue to obtain this flexibility and concurrent development is to have a unified information flow from early product and process design to the final control and operation of the manufacturing plant.

In [1] this issue is discussed and manufacturing operations and their relations, the sequences of operations (SOPs), are introduced. In [2] a formal model for operations is introduced based on extended finite automata (EFAs) [3] and all the information on the relations between the individual operations is then encapsulated in modularized EFA models. It is shown that product, process and resource design can be integrated via operations, and the relation among these operations can be graphically visualized by the Sequence Planner tool.

Based on the operation model, a method for automatic generation of controllers for collision-free Flexible Manufacturing Systems (FMSs) is introduced in [4]. In this method, for each operation in the system, a set of shapes (3D sweep volumes) is generated. Then, based on pairwise intersection tests over all the shapes, mutual exclusion zones are identified and avoided by adding guards in the corresponding operation models. The automatic generation also includes a synthesis procedure, where a non-blocking and controllable supervisor is generated, [5]. This supervisor is preferably synthesized by the tool Supremica [6], including a new approach proposed in [7] where the supervisor is generated by adding guards to the original EFA models. In this way the supervisor can be implemented by ordinary industrial programmable logic controllers.

In an FMS including a large number of robots and devices, the total amount of discrete states that need to be considered in the synthesis will increase dramatically, thus making it hard to compute a supervisor for industrially interesting cases. The aim of this paper is to reduce this complexity by introducing a method to efficiently represent SOPs for synthesis. A single EFA model for consecutive operations in the system will be generated rather than an EFA model for each operation. These EFAs will be further reduced by identifying local transitions without any interaction with other operations. Such transitions can be removed and related source and target states can be merged to reduce the total number of states in the EFAs that are used for supervisor synthesis.

The presented work is divided into four sections. In Section II mathematical preliminaries are shortly reviewed. Section III presents the reduction method in detail, followed by Section IV, where a case study is presented that illustrates the efficiency of the suggested method.

II. Preliminaries

This section presents an operation model based on *Extended finite automata* [3], together with a formal modeling language called *Sequences of operations* for self-contained and hierarchical operations, defined in [2]. Moreover, graph and flow networks are presented, to be used for identifying sequential relations among self-contained operations.

A. Operation model

Operations and the relations between them in terms of sequences and other conditions are modeled by extended finite automata, which are ordinary automata augmented with variables, guard formulas and action functions.

Definition 1 (Extended finite automaton)**.** An extended finite automaton (EFA) is a 7-tuple

$$E = \langle L \times V, \Sigma, \mathcal{G}, \mathcal{A}, \rightarrow, (\ell_0, v_0), M \rangle. \tag{1}$$

The set $L \times V$ is the extended finite set of states, where L is a finite set of *discrete locations* and V is the finite domain of an m-tuple of variables, $v = (v^1, v^2, \ldots, v^m)$. Σ is a nonempty finite set of events (the alphabet). $\mathcal{G}$ is a set of guard predicates over V, $\mathcal{A}$ is a set of action functions from V to V, where each function maps the present variable values to the variable values of the next state. $\rightarrow \subseteq L \times \Sigma \times \mathcal{G} \times \mathcal{A} \times L$ is a state transition relation, $(\ell_0, v_0) \in L \times V$ is the initial state, and

$M \in L \times V$ is a set of marked (desired) states. The notation $\ell \xrightarrow{\sigma}_{g/a} \ell'$ is used as shorthand for $(\ell, \sigma, g, a, \ell') \in \rightarrow$. □

It is assumed that the guards have been parsed and written in conjunctive normal form $g = g^1 \wedge \ldots \wedge g^j$, where each or-clause $g^i(v) = g^{i,1}(v^1) \vee \ldots \vee g^{i,m}(v^m),\ i = 1, \ldots, j$, compares the variables with a constant in $V = V^1 \times \cdots \times V^m$. We also assume that all actions are written as constant functions $v := a = (a^1, \ldots, a^m)$. Any transition can be decomposed into multiple transitions of this form. For instance, the transition $p \xrightarrow{\sigma}_{x:=y+1} q$ where $y \in \{0, 1\}$ can be decomposed into: $p \xrightarrow{\sigma}_{y=0/x:=1} q$ and $p \xrightarrow{\sigma}_{y=1/x:=2} q$. The symbol ξ is used to denote implicit actions that update variables to their current value, and in vector form $\Xi = (\xi, \ldots, \xi)$. Sometimes it is of interest to know the explicit state transition relation $\mapsto$, which is defined as

$$\begin{aligned} \mapsto := \{(\ell, v, \sigma, \ell', v') \in L \times V \times \Sigma \times L \times V \mid \\ \exists\ \ell \xrightarrow{\sigma}_{g/a} \ell' \text{ such that} \\ g(v) = 1 \text{ and } v'^i := \begin{cases} a^i & a^i \neq \xi \\ v^i & a^i = \xi \end{cases} \end{aligned} \tag{2}$$

EFAs are composed by full synchronous composition [8]. In the composition of two EFAs, a shared event is enabled if and only if it is enabled by each of the composed automata [3].

An operation can be formally modeled as an EFA. For notational purposes, the guards and actions are replaced by a set C of transition conditions in terms of predicates over the variables, including both their current and next values after a transition, c.f. [2].

Definition 2 (Operation)**.** An operation O_k is an EFA where the set of discrete locations $L_k = \{O_k^i, O_k^e, O_k^f\}$, the event set $\Sigma_k = \{O_k^{\uparrow}, O_k^{\downarrow}\}$, the set of transition conditions $C_k = \{C_k^{\uparrow}, C_k^{\downarrow}\}$, the transition relation $\rightarrow_k = \{(O_k^i, O_k^{\uparrow}/C_k^{\uparrow}, O_k^e), (O_k^e, O_k^{\downarrow}/C_k^{\downarrow}, O_k^f)\}$, the initial location $\ell_k^i = O_k^i$, and all locations are marked, $M = L$. □

In order to include requirements on the operation locations in the conditions, these locations are also represented as Boolean variables O_k^i, O_k^e and O_k^f, included in the variable set V. Each of these variables is equal to one when the corresponding location is active. In this paper, for the sake of simplicity and space, the actions that update the location variables are not explicitly shown on the transitions. Also, it is assumed that events in the system are local and controllable, i.e. there are no common events in operation models; communications are done by variables.

B. Operation sequences and hierarchical structures

When a number of operations are interacting, the basic assumption is that all operations are running in parallel. This is modeled by the full synchronous composition operator $||$. For a SOP SO with n operations $O_1, O_2, \ldots, O_n$ the composite operation model is defined as

$$SO = O_1 || O_2 || \cdots || O_n \tag{3}$$

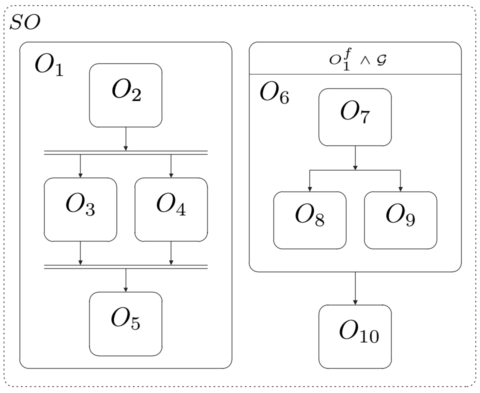

Fig. 1. Sequences of operations (SOP) in the running example. Here O_1^f means O_1 must be finished before O_6 can start and $\mathcal{G}$ is any boolean guard expression.

The SOP can be represented graphically with a set of sequences, using the graphical notations introduced in [2]. A sequence is a graph that connects a set of operations that are related to each other. The operation conditions are represented with arrows, lines and boolean expressions, see examples in Fig. 1. A sequence or a set of sequences of operations SO_k can be encapsulated by a hierarchical operation O_k executed concurrently with SO_k as

$$O_k || SO_k \tag{4}$$

The pre- and postconditions in O_k are then reformulated to synchronize with the desired start and complete operations in SO_k. Hierarchical operations including their lower level SOPs, cf. SO_k in (4), can be put together to generate a new SOP SO on the higher level.

$$SO = O_1 || SO_1 || \cdots || O_n || SO_n \tag{5}$$

In reality all models are running in parallel, but from a user point of view the SO_k on one level are encapsulated as hierarchical operations on the next higher level. To enforce that a number of SOPs $SO_1, \ldots, SO_n$ are executed, a top-level hierarchical operation O combining (4) and (5) is introduced, where the postcondition $C^{\downarrow} = O_1^f \wedge \cdots \wedge O_n^f$ and only the final location O^f is marked. This guarantees that all SOPs will be completed, where this final specification step is important e.g. for supervisory synthesis of nonblocking supervisors. Formal tools for EFAs are available, cf. [9]. Therefore, it is possible to formally verify and synthesize a supervisor by the suggested SOP language.

C. Graph and flow network

A *graph* is a structure $G = (V, E)$ in which V is a set of *vertices* and E is a set of binary relations between the vertices called *edges*. A *(directed) finite path* in G is a non-empty sequence of vertices $v_1, v_2, \cdots v_m \in V$ such that for any two consecutive vertices v_i and v_{i+1}, there is an edge $(v_i, v_{i+1}) \in E$ and $v_1, v_2, \cdots v_m$ are all different. An $s - t$ path is a finite path that is started in the source $v_1 = s$, and finished in the terminal $v_m = t$.

Definition 3 (Independent $s-t$ paths)**.** Two $s-t$ paths are said to be independent if they do not have any vertices in common apart from s and t. □

Definition 4 (Flow network)**.** Let $N = (V, E)$ be a directed graph and let $c : E \longrightarrow \mathbb{R}^+$ be a *capacity function* and $\{s, t\} \in V$. A function $f : V \times V \longrightarrow \mathbb{R}$ is called a *flow* if:

$$f(u, v) \leq c(u, v) \qquad \forall \ \{u, v\} \in V \tag{6}$$

$$f(u, v) = -f(v, u) \qquad \forall \ \{u, v\} \in V \tag{7}$$

$$\sum_{e \in \delta^{in}(v)} f(e) - \sum_{e \in \delta^{out}(v)} f(e) = 0 \qquad \forall \ v \in V \backslash \{s, t\} \tag{8}$$

Here $\delta^{in}(v)$ and $\delta^{out}(v)$ denote the set of incoming and outgoing edges to and from v, respectively. This graph including the flow is called a flow network. □

The first condition (6) is the *Capacity constraints*: the flow in an edge cannot exceed its capacity, the second condition (7) is the *Skew symmetry* and the last condition (8) is the *Flow conservation law*: the amount of flow entering a vertex $v \neq \{s, t\}$ should be equal to the amount of flow going out from v. In the *maximum-flow problem*, we are given a flow network G with source s and sink t, and we wish to find the maximum flow in the network.

III. Reduced-order synthesis

When a number of operations are interacting, the basic assumption is that all operations are running in parallel. This is modeled by the SOP SO_k in (3), where all the restrictions are represented by pre- and postconditions included in the individual operation models. However, for a large system the total amount of discrete states that need to be considered in the synthesis will increase dramatically.

This section presents a method to automatically identify the sequences among a set of individual operation models and simplify them into one EFA. Then, these simplified EFAs will be further reduced by identifying the local transitions in each EFA. The reduced-order models are used for synthesis of a supervisor that generates a set of guards that gives a nonblocking and maximal permissive closed loop system.

A. Sequential relation graph

The sequential restrictions on the order between different operations are formally expressed by logical preconditions. To find the sequential relation between operations in a SOP, Algorithm 1 is used to generate a sequential relation graph, i.e. a graph in which vertices are the operations and the edges are the binary sequence relation between each pair of operations.

In Algorithm 1, the function *createVertices* creates vertices in the graph G based on the operations in a given SOP. Here, the injective function $\vartheta : O \rightarrowtail V$ maps an operation to a vertex in the graph G. The function *createEdges* will go through all guards in each operation and identify the operations which have a sequential relation to the current one and add an edge between the corresponding vertices.

In order to find a set of independent paths covering all operations, the graph G is converted to an extended flow network N in which all the edges and vertices has the capacity equals to one and calculates the flow value $f(e)$ for each edge to maximize the flow in the network. Since f is an integer flow and all capacities are 1, the flow value $f(e)$ will be 0 or 1 for all edges in the network. Therefore, the independent paths in G will be found by considering $s - t$ paths in the network where all edges have flow value equal to one. Such paths are obtained by the flow maximization. Maximum flow in a network is a well known problem in graph theory which can be solved by the Edmonds-Karp algorithm [11] that is an implementation of the Ford-Fulkerson method. [12].

Algorithm 1 Sequential relation graph

```
    GENERATE-GRAPH($SOP$)
1:  $G = (V, E)$
2:  createVertices($V[G], SOP$)
3:  createEdges($E[G], SOP$)
4:  return $G$

    CREATE-VERTICES($V[G], SOP$)
1:  foreach $O_m \in SOP$ do
2:    $V[G] \leftarrow V[G] \cup \vartheta(O_m)$
3:  end for
4:  return

    CREATE-EDGES($E[G], SOP$)
1:  foreach $O_m \in SOP$ do
2:    $v_m \leftarrow \vartheta(O_m)$
3:    foreach $g^j_m(v) \in C^{\uparrow}_m$ do
4:      if $|g^j_m(v)| = 1$ then
5:        $g^{j,1}_m(v^1) \leftarrow$ getFirstElement($g^j_m(v)$)
6:        if $g^{j,1}_m(v^1)$ is a type of $O^f$ then
7:          $O_n \leftarrow$ getRelatedOperation($g^{j,1}_m(v^1)$)
8:          $v_n \leftarrow \vartheta(O_n)$
9:          $E[G] \leftarrow E[G] \cup \{(v_n, v_m)\}$
10:       end if
11:     end if
12:   end for
13: end for
14: return
```

Definition 5. Let p be a path in graph G, then the functions $V(p)$ and $E(p)$ return the set of vertices and edges on the path p, receptively. □

Proposition 1. *Let N be a network with maximum flow and $c(e) = 1$ for all edges extended with $c(v) = 1$ for all vertices. Then the $s-t$ path p_i in N is independent if for all $e \in E(p_i)$ we have $f(e) = 1$.*

Proof: To remove the capacity on the vertices, the net N is expanded such that each $v \in V[N]$ is replaced by v_{in} and v_{out}, where v_{in} is connected by edges going into v and v_{out} is connected to edges coming out from v. Then assign capacity $c(v)$ to the edge connecting v_{in} and v_{out}. This means that the expanded network has the same behavior as the original one. Furthermore, it can be treated as an ordinary maximum flow

Algorithm 2 Maximum-flow in a graph

```
    INDEPENDENT-PATHS($G$)
 1: $P \leftarrow \emptyset$
 2: while $\exists\ v \in V[G]$ do
 3:     $P_t \leftarrow \emptyset$
 4:     $N \leftarrow$ Convert-To-Flow-Network($G$)
 5:     Edmonds-Karp($N$)
 6:     $P_t \leftarrow$ getAllIndependentPaths($N$)
 7:     $P \leftarrow P_t$
 8:     $V[G] \leftarrow V[G] \setminus \{V(p_{ti}) : p_{ti} \in P_t, s, t\}$
 9: end while
10: return $P$

    CONVERT-TO-FLOW-NETWORK($G$)
 1: $N(V, E) \leftarrow G(V, E)$
 2: $V[N] \leftarrow V[N] \cup \{s, t\}$
 3: foreach $v_i \in V[N]$ do
 4:     $c(v_i) \leftarrow 1$
 5:     if $v_i$ has no incoming edge then
 6:         $E[N] \leftarrow E[N] \cup \{(s, v_i)\}$
 7:         $c(s, v_i) \leftarrow 1$
 8:     else if $v_i$ has no outgoing edge then
 9:         $E[N] \leftarrow E[N] \cup \{(v_i, t)\}$
10:         $c(v_i, t) \leftarrow 1$
11:     end if
12: end for
13: return $N$
```

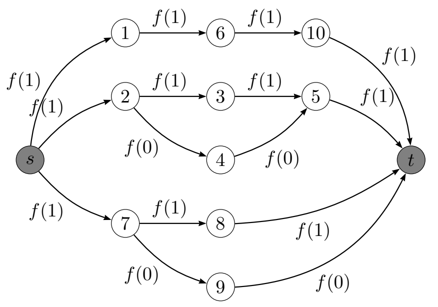

Fig. 2. Flow network of the first iteration on the example SOP. Here, s and t are the source and sink vertices and i in $f(i)$ is the calculated flow value.

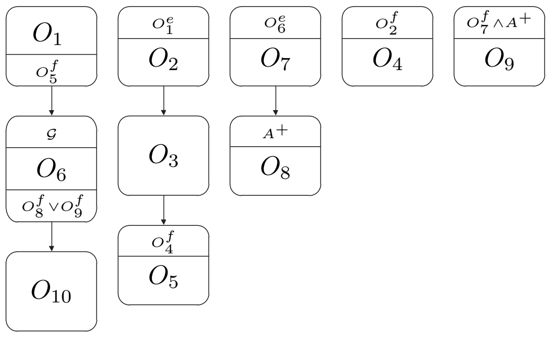

Fig. 3. Independent sequences of the operations for the SOP in Fig. 1. Here the alternative operations O_8 and O_9 are modeled by a mutual exclusion variable A, where A^+ means $A = 0/A := 1$, c.f. [2].

problem without capacity constrains on the vertices, cf. [13]. Since f is an integer flow and all capacities are 1, $f(e) \in \{0, 1\}$ for all $e \in E$. Because every vertex v_{in} has only one outgoing edge, at most one incoming edge e of v_{in} can have $f(e) = 1$, c.f. (8). Similarly, because every vertex v_{out} has only one incoming edge, at most one outgoing edge e of v_{out} can have $f(e) = 1$. Therefore, the independent paths p_i in N are the $s - t$ paths where all edges $e \in E(p_i)$ have the flow value equal to 1. ■

Proposition 2. *Let G be the sequential relation graph of a SOP SO. Then each independent path in G represents an independent sequence of operations in SO, i.e. sequences of operations that do not have any operations in common.*

Proof: Let P be the set of independent paths p_i in G and $\vartheta : O \rightarrowtail V$ be an injective function that maps an operation to a vertex in the graph G, then from Definition 3 and 5 we have:

$$\bigcap_{p_i \in P} (V(p_i) \setminus \{s, t\}) = \emptyset$$

therefore,

$$\bigcap_{p_i \in P} \vartheta^{-1}(V(p_i) \setminus \{s, t\}) = \emptyset$$

which means that there are no common operations in any sequence of operations. ■

Algorithm 2 iteratively find the independent paths of the input graph G. The function *Convert-To-Flow-Network* converts the graph G to the corresponding network N and assigns capacity one for all edges and vertices. Two super nodes s and t will be added to the network with the infinity in and out flow capacity, and all the vertices with no incoming and outgoing edge will be connected to the nodes s and t, respectively.

Next, the *Edmonds-Karp* function first converts the network N to a normal flow network without capacity on vertices. Then the maximum-flow is calculated and the flow value of each edge is updated to maximize the flow in the network. Furthermore, the function *getAllIndependentPaths*, will find the $s - t$ paths p_{ti} in G by following the edges with the flow capacity equal to 1. These paths, excluding the s and t vertices, are added to the path set P and then removed from the graph G. The algorithm will iterate on the graph and terminate when there are no more vertices. The output of the algorithm then will be the path set P, which contains all the possible independent paths in graph G.

The network generated in the first iteration for the SOP in Fig. 1 including the flow value for each edge is depicted in Fig. 2. The set of independent paths after four iterations will be $P = \{\langle 2 \rightarrow 3 \rightarrow 5\rangle, \langle 1 \rightarrow 6 \rightarrow 10\rangle, \langle 7 \rightarrow 8\rangle, \langle 9\rangle, \langle 4\rangle\}$. The corresponding sequences of operations of the independent paths in the path set P are illustrated in Fig. 3. Observe that, for instance, the operations O_8 and O_9 in Fig. 1 executed in arbitrary order and in Fig. 3 in mutual exclusion by the common resource R.

B. Reduced-Order EFA models

Sequential relations among operations are introduced as preconditions in the modular EFA operation models. In manufacturing systems most of the operations are organized into sequences, and therefore most of the operation conditions are based on this relationship. In this part, first the operation models are simplified by generating one EFA for each independent sequence. The conditions related to sequential relations are then removed. For instance, for the operation O_5 preceded by O_3 in Fig. 3 with the transition conditions $C_5^{\uparrow} = O_4^f \wedge O_3^f$, after simplifying into one synchronized EFA model, the condition becomes $C_5^{\uparrow} = O_4^f$.

In the next step, some of the transitions will be identified as local with respect to other EFAs in the system. These transitions will have no impact in the synthesis procedure. Hence, they can be removed and a reduced order model will be achieved.

However, it needs to be shown that the supervisor before and after reducing the order has the same behavior in terms of supervisor guards. Remind that the supervisor is generated by adding guards to the original EFA models, see further details in Section III-C. Now we consider the case when EFA E_1 will be reduced, and the other EFAs are synchronized to EFA E_2. We also repeat that communication between the EFAs is performed by guards and actions; no shared, only local events are assumed. Furthermore, assume without loss of generality that the tuple of variables for E_1 and E_2 is common and denoted v.

Definition 6 (Local and global variables)**.** Let E_1 and E_2 be two EFAs including the common tuple of variables $v = (v_1, v_2, v_g)$, where v_1 is only updated in E_1, v_2 is only updated in E_2, and v_g is the tuple of shared variables updated in both E_1 and E_2. Then the variables in v_1 and v_2 are *local*, while the variables in v_g are *global*. Similarly, the action set a is partitioned as $a = (a_1, a_2, a_g)$, where $v_k := a_k,\ k = \{1, 2, g\}$. □

Definition 7 (Local transition)**.** Let E_1 and E_2 be two EFAs. Then a transition $(\ell_1, \sigma, g, a, \ell_1') \in\to_1$ is a *local transition* in E_1 if $g \vdash \texttt{true}$, $a = (a_1, a_2, a_g)$ is restricted to $a_2 = \Xi$, and $a_g = \Xi$. □

This means that any local transition in E_1 can occur at any time, independently of E_2.

Definition 8 (Guard invariant transition)**.** Let E_1 and E_2 be two EFAs. Then the transition $(\ell_1, \sigma, g, a, \ell_1') \in\to_1$, alternatively expressed as the explicit transition $(\ell_1, v_1, v_2, v_g) \overset{\sigma}{\mapsto} (\ell_1', v_1', v_2, v_g)$, is *guard invariant* if $g_2(v_1, v_2, v_g) = g_2(v_1', v_2, v_g)$ for all $g_2 \in \mathcal{G}_2$ in E_2 and arbitrary (v_2, v_g). □

A guard invariant transition in E_1 implies there are no guards in E_2 depending on any variable updated in the guard invariant transition.

Definition 9 (Solitary outgoing local transition (SOLT))**.** Let E_1 and E_2 be two EFAs. Then a local transition $(\ell_1, \sigma, g, a, \ell_1') \in\to_1$ is a *solitary outgoing local transition,* if ℓ_1 has only one outgoing transition, $\ell_1' \neq \ell_1$, and for all outgoing transitions from ℓ_1', the related guards have no condition on the updated variables in a and the related actions update the same variables as in a. □

The last condition means that the outcomes of the guards at transitions after a SOLT do not depend on the execution of the SOLT action a. This restriction is crucial to be able to remove a SOLT.

Lemma 1. *Let E_1 and E_2 be two EFAs with common variables $v = (v_1, v_2, v_g)$. If there exists a guard invariant SOLT $(\ell_1, \sigma, g, a, \ell_1') \in\to_1$, then E_1 can be reduced as*

$$\begin{aligned}\to_1 &= \to_1 \setminus \{(\ell_1, \sigma, g, a, \ell_1')\}\\ &\cup\ \{(\ell_1, \bar{\sigma}, g', a^r, \ell_1'') \mid (\ell_1', \bar{\sigma}, g', a', \ell_1'') \in\to_1\}\end{aligned}$$

where

$$a^{ri} = \begin{cases} a'^i & \text{if } a'^i \neq \xi \\ a^i & \text{otherwise} \end{cases};$$

$$M_1 = \begin{cases} M_1 \cup \{\ell_1\} & \text{if } \ell_1' \in M_1 \\ M_1 & \text{if } \ell_1' \notin M_1; \end{cases}$$

$$\Sigma_1 = \Sigma_1 \setminus \{\sigma\}.$$

The reduced EFA is denoted E_1^r and the language $\mathcal{L}(E_1^r || E_2) = P_{\Sigma\setminus\sigma}(\mathcal{L}(E_1 || E_2))$, where $P_{\Sigma\setminus\sigma}$ is the projection where σ is removed from all strings in the corresponding language.

Proof: Introducing the notations $q_\sigma = (\ell_1, \ell_2, v_1, v_2, v_g)$ and $q_\sigma' = (\ell_1', \ell_2, v_1', v_2, v_g)$, where $\ell_k \in L_k,\ k = 1, 2$, and $Q = L_1 \times L_2 \times V$ as well as $\Sigma = \Sigma_1 \cup \Sigma_2$, the set of reachable source states in $E_1 || E_2$, corresponding to the SOLT $(\ell_1, \sigma, g, a, \ell_1') \in\to_1$, is defined as

$$Q_\sigma = (q_\sigma \in Q | s\sigma \in \mathcal{L}(E_1||E_2) \wedge q_\sigma \overset{\sigma}{\mapsto} q_\sigma' \wedge s \in \Sigma^* \wedge \sigma \in \Sigma_1)$$

Generally, Q_σ is a set of states since the location ℓ_1 is often reached in combination with a number of different locations in E_2 and/or values of v. The related set of target states can be expressed as

$$Q_\sigma' = (q_\sigma' \in Q | s\sigma \in \mathcal{L}(E_1||E_2) \wedge q_\sigma \overset{\sigma}{\mapsto} q_\sigma' \wedge s \in \Sigma^* \wedge \sigma \in \Sigma_1)$$

Observe that for each source state $q_\sigma \in Q_\sigma$ there is only one unique target state $q_\sigma' \in Q_\sigma'$, since the transition is a SOLT.

After a possible set of local transitions in E_2, resulting in different strings of events s', one of the outgoing transitions from ℓ_1' in E_1 is executed with the event $\bar{\sigma}$. Using the notations $q_{\bar{\sigma}} = (\ell_1', \ell_2', v_1', v_2', v_g')$ and $q_{\bar{\sigma}}' = (\ell_1'', \ell_2', v_1'', v_2', v_g'')$, the set of reachable target states in $E_1 || E_2$ after this transition, can be formulated as

$$\begin{aligned}Q_{\bar{\sigma}}' = (q_{\bar{\sigma}}' \in Q | s\sigma s'\bar{\sigma} \in \mathcal{L}(E_1||E_2) \wedge q_{\bar{\sigma}} \overset{\bar{\sigma}}{\mapsto} q_{\bar{\sigma}}'\\ \wedge\, s \in \Sigma^* \wedge s' \in \Sigma_2^* \wedge \sigma, \bar{\sigma} \in \Sigma_1)\end{aligned}$$

Due to the guard invariant SOLT, the corresponding set of source states $Q_{\bar{\sigma}}$ is indeed equal to the set of target states Q_σ'.

Algorithm 3 Reduced-order EFAs generation

```
    REDUCE-EFAs ($SOP, P$)
 1: $E \leftarrow \emptyset$
 2: while $\exists p_j \in P$ do
 3:    $E_i \leftarrow \emptyset$
 4:    foreach $v \in V[p_j]$ do
 5:       $O_v \leftarrow \vartheta^{-1}(v)$
 6:       $E_j \leftarrow E_j || O_v$
 7:    end for
 8:    foreach $(\ell, \sigma, g, a, \ell') \in \rightarrow_j$ do
 9:       if $(\ell, \sigma, g, a, \ell')$ is a guard invariant SOLT then
10:          $\rightarrow_j = \rightarrow_j \setminus \{(\ell, \sigma, g, a, \ell')\}$
             $\cup \{(\ell, \bar{\sigma}, g', a^r, \ell'') | (\ell', \bar{\sigma}, g', a', \ell'') \in \rightarrow_j\}$
11:          $a^r = a$
12:          foreach $a'^i \in a'$ do
13:             if $(a'^i \neq \xi)$ then $a^{ri} = a'^i$ end if
14:          end for
15:          if $(\ell'_1 \in M_1)$ then $M_1 \cup \{\ell_1\}$ end if
16:          $\Sigma_i = \Sigma_i \setminus \{\sigma\}$
17:       end if
18:    end for
19:    $E \leftarrow E_i$
20: end while
21: return $E$
```

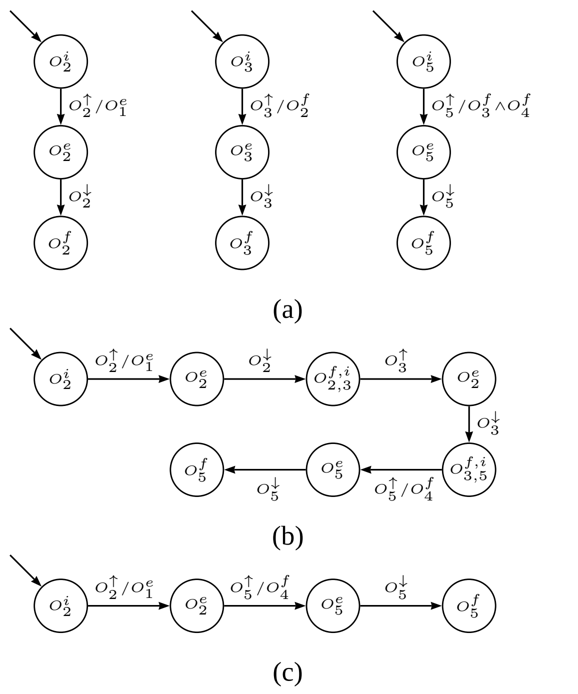

Fig. 4. The EFA models for the operations O_2, O_3 and O_5 in the path $\langle 2 \rightarrow 3 \rightarrow 5 \rangle$, (b) simplified EFA $E = O_2||O_3||O_5$ and (c) the reduced-order EFA E^r.

In the reduced system $E_1^r||E_2$ the same set of states Q_σ can be reached as in $E_1||E_2$, since the two systems have the same behavior before the event σ has been executed in $E_1||E_2$. Since the SOLT $(\ell_1, \sigma, g, a, \ell'_1) \in \rightarrow_1$ according to Definition 9 has only one local outgoing transition, and the outcomes of the guards at transitions after this SOLT do not depend on the action a, and furthermore the SOLT is also guard invariant, the same set of reachable target states $Q'_{\bar{\sigma}}$ after the event $\bar{\sigma}$ are achieved for both $E_1^r||E_2$ and $E_1||E_2$.

In total this means that the reachable set of states for the reduced system $Q_{E_1^r||E_2} = Q_{E_1||E_2} \setminus Q'_\sigma$. The only difference between the languages of the two systems is that the SOLT event σ is not included in $\mathcal{L}(E_1^r||E_2)$, meaning that $\mathcal{L}(E_1^r||E_2) = P_{\Sigma\setminus\sigma}(\mathcal{L}(E_1||E_2))$. ■

Definition 10 (Supervisor guards)**.** The function $Sup_g(E)$ generates a set of guards that gives a nonblocking and maximal permissive closed-loop system for the EFA E. □

Theorem 1. *Consider the synchronous system $E_1||E_2$ and the reduced synchronous system $E_1^r||E_2$. These two synchronous systems generate the same set of supervisor guards i.e.*

$$Sup_g(E_1||E_2) = Sup_g(E_1^r||E_2)$$

Proof: Based on Lemma 1 it is enough to show that no supervisor guard needs to be included at the transition $q_\sigma \overset{\sigma}{\hookrightarrow} q'_\sigma$ in $E_1||E_2$. Assume that a marked state is reachable from q'_σ. Then it is also always reachable from q_σ, since the transition $q_\sigma \overset{\sigma}{\hookrightarrow} q'_\sigma$ is generated by a SOLT in E_1 according to Definition 9. Hence, neither q_σ nor q'_σ is forbidden. On the contrary, if a marked state is not reachable from q'_σ, it is also never reachable from q_σ. The reason is again that the transition from q_σ to q'_σ is generated by a SOLT in E_1. Hence, both q_σ and q'_σ are blocking states and therefore forbidden. This means that either both q_σ and q'_σ are allowed or both are forbidden, implying that no supervisor guard needs to be included at the transition $q_\sigma \overset{\sigma}{\hookrightarrow} q'_\sigma$.

Since the behavior of $E_1^r||E_2$, with the exception of this transition, is the same as $E_1||E_2$, according to Lemma 1, we finally conclude that the guards generated by $Sup_g(E_1||E_2)$ are the same as the guards generated by $Sup_g(E_1^r||E_2)$. ■

Since the supervisor guards generated by $Sup_g(E_1^r||E_2)$ appear at the transitions also existing in $E_1||E_2$, the generated guards on E_1^r can be transferred back to the original EFA E_1.

Algorithm 3 reduces the possible transitions according to Theorem 1. First the models are simplified into one synchronized EFA model and then further reduced by iterating over transitions and removing the transitions that has no impact on the synthesis procedure. In Fig. 4, (a) illustrates the EFA models for the operations O_2, O_3, and O_5 in the path $(2 \rightarrow 3 \rightarrow 5)$, (b) the EFA $E = O_2||O_3||O_5$, and (c) the reduced-order EFA E^r. The transitions $(O_2^e, O_2^{\downarrow}, \texttt{true}, \Xi, O_{2,3}^{f,i})$, $(O_{2,3}^{f,i}, O_3^{\uparrow}, \texttt{true}, \Xi, O_3^e)$, and $(O_3^e, O_3^{\downarrow}, \texttt{true}, \Xi, O_{3,5}^{f,i})$ are removed and the locations O_2^e and O_5^e are connected by the transition $(O_2^e, O_5^{\downarrow}, O_4^f, \Xi, O_5^e)$. Observe that the remaining transitions are not removed since each transition either has a guard or is not guard invariant. Reduced-order EFA models for the actual operations are generated based on Algorithm 1-3, and a non-blocking and maximally permissive supervisor [5] can be computed.

TABLE I
COMPARISON OF TWO METHODS

Method	$\|SOP\|$	$\|EFA\|$	$\|Q_{reach}\|$	$\|Q_{sup}\|$	$\|BDD\|$	$\|Sup_g\|$	Time(s)
Normal	82	164	4.023620354×10^9	1.758696194×10^9	886	4	78
Efficient	82	64	5.235889×10^6	2.961409×10^6	262	4	2

C. Synthesis

Traditionally, the synthesis procedure for EFAs has been carried out by first flattening the EFAs to ordinary finite automata, and then perform a monolithic synthesis. In this way, the states of the final supervisor are represented explicitly, which has some main drawbacks when the supervisor becomes very large in terms of the number of states [4].

In [7], a framework is presented, where the user model a system by EFAs and obtain the supervisor modularly in form of EFAs. The only difference between the original and synthesized supervisor EFAs is that the guards are extended in the latter model. The main advantage of this approach is that the resulting supervisor, represented as modular EFAs, becomes more comprehensible for the user, and the supervisor can easily be implemented in an industrial controller. In addition, the user will remain in the same model domain, i.e. EFAs, both when models are introduced and supervisors are generated.

To be able to handle large systems, all computations based on the reduced EFAs are performed symbolically using Binary Decision Diagrams (BDDs). The procedure is carried out in five main steps. Initially, the EFAs are converted to BDDs. Based on the BDDs the supervisor is computed, which is then used to extract the guards. To make the guards more tractable for the users, the guards are then reduced by some heuristic techniques. Finally, the reduced guards are attached to the original EFAs. This procedure can be repeated iteratively, making it possible for the user to perform further modifications on the obtained supervisor and then compute a new supervisor. The details of this approach is beyond the scope of this paper. For a more detailed elaboration of the guard generation procedure refer to [7].

IV. CASE STUDY

The method described above has been implemented as a toolbox in Sequence Planner software and employed to efficiently generate the EFA models for a given SOP modeled by user. To demonstrate the efficiency of the method, we have modeled an extended version of a robot cell at Chalmers Robot and Automation Lab, using Sequence Planner and the toolbox to generate the reduced-order EFA models. The extended cell consists of five ABB robots, two fixtures, an AGV, and a conveyor. The desired behavior of the cell is the following: Two parts are loaded by the operator and conveyed to a robot station by the conveyor. Two robots pick and place the parts on a fixture, where they are fixated by the clamps. Then, the robots assemble the parts by drilling and pop-riveting the predefined geometry points. After that, the assembled parts are unloaded by the third robot and delivered to a second station for more pop-riveting. In that station, two other robots pop-rivet the remaining points, and then the finished product is unloaded by the third robot and finally transported from the workstation by an AGV.

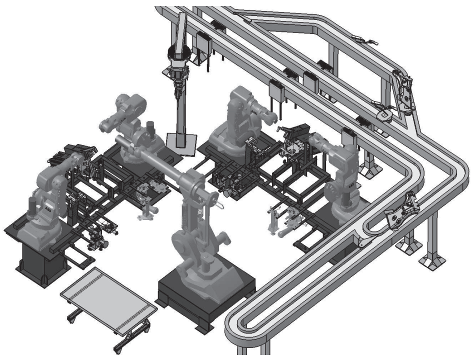
Fig. 5. 3D simulation of the robot cell in the case study.

The cell is simulated in 3D simulation software Dassault Systèmes DELMIA V5, see Fig. 5, and in order to automatically avoid collision possibilities among devices the method introduced in [4] is applied where a set of shapes (3D volumes of the device model) are created for all operations. Based on pairwise intersection tests over these shapes, the volumes where the robots and other devices may collide, are identified. To avoid collision possibilities, necessary guards are added to the corresponding operation models. Furthermore, reduced-order models of the operations are generated according to Section III-B, including collision avoidance guards, and a non-blocking supervisor is achieved using the Supremica software.

For the mentioned cell, 242 shapes are created where among them 117 pairs of intersected shapes are identified. To avoid collisions in these mutual exclusion zones, 234 guards are automatically added to the operation models in Sequence Planner.

Table I shows statistics of the cell and the supervisor based on the reduced-order EFA model generated by the presented method, and the general operation model defined in Definition 2. Here, $|SOP|$ is the number of operations, $|EFA|$ is the number of generated EFAs for the given SOP, $|Q_{reach}|$ and $|Q_{sup}|$ represents the number of reachable states in the closed-loop model and supervisor, respectively, $|BDD|$ is the number of BDD variables used to model the system and $|Sup_g|$ is the number of generated guards for the supervisor to be non-blocking. The table also includes the time for computing the supervisor. In Table I, the number of guards generated by the supervisor for the two methods are equal, which implies these two supervisors generate the same closed loop behavior.

V. Conclusions

The state space explosion in a large manufacturing system with detailed and complex sequences of operations is inevitable. To approach this problem, a method reduce the order of the operation models for synthesis has been presented in this paper. This is done by finding the operations with sequential behavior which are identified by mapping the operations in a graph and finding the independent paths in the corresponding network. Instead of presenting each operation with an three-location EFA, a sequence of operations is represented efficiently as a single EFA model in which the transitions that have no impact on the synthesis procedure are eliminated.

A case study is presented where a large manufacturing cell with five robots, fixtures and a conveyor is efficiently modeled. The reduction principle is applied to the synthesis of non-blocking operation sequences, where collisions among moving devices are guaranteed to be avoided by a flexible booking process. It is shown that the number of states in the synthesis procedure and the computation time are reduced dramatically by the suggested method. It is also proved that the controller, before and after applying the reduction method, generates the same closed loop behavior.

VI. Acknowledgement

This work was carried out at the Wingquist Laboratory VINN Excellence Center within the Area of Advance – Production at Chalmers, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA). The support is gratefully acknowledged.

References

[1] K. Bengtsson, B. Lennartson, and C. Yuan, “The origin of operations: Interactions between the product and the manufacturing automation control system,” in *13th IFAC Symposium on Information Control Problems in Manufacturing Technology, INCOM09, Moscow, Russia*, Moscow, Russia, Jun. 2009, pp. 40–45.

[2] B. Lennartson, K. Bengtsson, C. Yuan, K. Andersson, M. Fabian, P. Falkman, and K. Åkesson, “Sequence planning for integrated product, process and automation design,” *Automation Science and Engineering, IEEE Transactions on*, vol. 7, no. 4, pp. 791–802, Oct. 2010.

[3] M. Sköldstam, K. Åkesson, and M. Fabian, “Modeling of discrete event systems using finite automata with variables,” *Decision and Control, 2007 46th IEEE Conference on*, pp. 3387–3392, 2007.

[4] M. R. Shoaei, B. Lennartson, and S. Miremadi, “Automatic generation of controllers for collision-free flexible manufacturing systems,” in *6th IEEE International Conference on Automation Science and Engineering*. IEEE, Aug. 2010, pp. 368–373.

[5] P. J. G. Ramadge and W. M. Wonham, “The control of discrete event systems,” *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[6] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, “Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems,” in *Proceedings of the 8th international Workshop on Discrete Event Systems, WODES’08*, Ann Arbor, MI, USA, 2006, pp. 384–385.

[7] S. Miremadi, B. Lennartson, and K. Åkesson, “A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata,” *accepted for IEEE Transactions on Control Systems Technology*, 2011.

[8] C. A. R. Hoare, *Communicating sequential processes*, ser. Series in Computer Science. ACM, Aug. 1978, vol. 21, no. 8.

[9] “Supremica.” [Online]. Available: htto://www.supremica.org

[10] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, “Compositional Synthesis of Maximally Permissive Supervisors Using Supervision Equivalence,” *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, Aug. 2007.

[11] J. Edmonds and R. M. Karp, “Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems,” *J. ACM*, vol. 19, no. 2, pp. 248–264, Apr. 1972.

[12] L. Ford and D. Fulkerson, “Flows in networks,” 1962.

[13] J. Wang and J. Silvester, “Maximum number of independent paths and radio connectivity,” *IEEE Transactions on Communications*, vol. 41, no. 10, pp. 1482–1494, 1993.