Deadlock avoidance for multi product manufacturing systems modeled as sequences of operations

Deadlock avoidance for multi product manufacturing systems modeled as sequences of operations

Patrik Bergagård and Martin Fabian
Department of Signals and Systems Chalmers University of Technology
{patrikm, fabian}@chalmers.se

***Abstract*— This paper demonstrates how industrial models of the sequential order between manufacturing operations may be connected with academic resource deadlock avoidance policies. Given a set of product types each modeled as a sequence of operations, the operations can be adapted to control concurrent manufacturing of multiple product instances, without resource deadlocks and with maximal flexibility. The given sequences of operations may contain routing flexibility, conjunctive and/or disjunctive resource requirements, and product assembly. Using a known deadlock avoidance approach, this paper presents how to transform the given operations into models required for the deadlock avoidance approach and how the calculated policy may be retransformed back into the operations.**

I. Introduction

Operations and *sequences of operations* are common elements that interconnect products, processes, resources, and automation information during design of products and manufacturing systems in industry [1]. Machining and assembly of products can be described by operations where each operation requires one or multiple resources from a manufacturing system. In order to raise the resource utilization, multi product manufacturing systems are designed to process more than one product at the same time. The products can be instances from a single product type or from multiple product types.

It is a challenging task to design a flexible control function for these types of systems. One challenge has to do with allocation of resources for the different products. It is well known that concurrent processing of products in systems with finite resource capacity may result in deadlocks [2]. A common industrial solution to these type of problems are over-specified control functions that limit the flexibility of the manufacturing system [3]. Well defined mathematical methods, i.e. formal methods, are seldom used.

To find deadlock avoidance control policies is a well established problem in the discrete event systems community. The systems are classified as different types of *Resource Allocation Systems* (*RASs*), for which different policies are valid [4]. One way to calculate a control policy is to use the synthesis algorithm in the *supervisory control theory* [5]. Synthesizing an optimal supervisor belongs to the class of NP-complete problems [6]. Some work has therefore aimed for sub-optimal solutions that are computationally efficient, see for example [7], but these solutions are also less flexible. Others have aimed for the optimal solution, for example [8].

This work has been carried out at the Wingquist Laboratory VINN Excellence Centre within the Production Area of Advance at Chalmers. It has been supported by the European 7th FP, grant agreement number 213734 (FLEXA) and Vinnova. The support is gratefully acknowledged.

The control policy can be developed *online* or *offline*. An online policy can be recalculated, based on the current products in the system, when a product enters or exits. The benefit is a limited state space in the calculation. The drawback is the need for extra algorithms online. An approach is presented in Papers IV and V in [9], where each new product that enters is instantiated from a product type library. An offline policy copes with all product combinations in advance. The offline deadlock avoidance policy can be implemented into the control function, therefore no extra online algorithms are needed. A drawback is the possible state space explosion during the offline calculation.

Many of the approaches presented in the literature have focused on deadlock avoidance policies for products that undergo different machining sequences in the same system, see for example [10], [8], [7]. Deadlock avoidance policies for systems that include assembly is less common, see for example [11] that use Petri nets for modeling and [12] that uses digraphs.

Motivated by the above observations, this paper tries to bridge the operation models from industry with academic deadlock avoidance policies. This paper describes an offline approach for how to adapt sequences of operations, that model processing of single product instances in multi product systems with reusable resources, to operations that can control processing of multiple product instances. The adapted operations are applied with the avoidance policy presented in [10] to enable concurrent processing without deadlocks. The focus in this paper is on the bridging of operation models and deadlock policies, thus any policy could have been used for demonstration. The approach in [10] is selected since the models are condensed and the approach aims for the optimal, most flexible, avoidance policy.

Moreover, the presented approach allows a user to specify which resource(s) each operation requires and not have to explicitly deal with resource allocation and deallocation. The requirement can be a conjunction and/or a disjunction of resources. More than one instance of a resource type can be required. The sequences of operations may include routing alternatives and product assembly.

II. Preliminaries

In this paper, a *manufacturing system* contains a set of *resource types*. The number of instances for a resource type X is denoted X^c.

Processes and tasks that are to be executed on a product are modeled by *operations*. The set of operations that are to be executed on each instance of a product type is given as a *sequence of operations* (*SOP*) [1]. Here, the term sequence does not necessarily imply a straight sequence, alternative and parallel execution are also allowed. A SOP is a generic model used in various types of industries, such as, batch processing, automotive, and manufacturing. An operation in a SOP can execute once for each product instance. An operation is supplemented with what resource type(s) that is(are) required in order to execute.

In this paper, each operation is appended with a *pre-* and/or *post- transition condition* that has to be satisfied before the operation can start and finish, respectively [1]. A *transition condition* (in the following just *condition*) can be used to put constraints on the current system state and on the succeeding system state.

In this paper, the sequential relations between operations in a SOP are modeled as conditions. The sequential relations are visualized with the graphical language Sequences of Operations, introduced in [1]. Figure 1 shows two SOPs. The syntax is exemplified on operation O13. O13 has a precondition. O13 can start if O12 has finished and that there is one instance of resource type M3 available in the system. The sequential relation to O12 is represented with an arrow and the resource requirement is given to the right of the operation label.

To formally model SOPs for calculating deadlock avoidance policies, *Extended Finite Automata* (*EFA*s) [13] are used. An (deterministic) EFA is a Finite Automaton with bounded integer variables. Each transition can be appended with a set of *guard expressions* (in the following just *guards*) and/or a set of *action functions* (in the following just *actions*). A transition is enabled if all guards evaluate to true and each action evaluates to a value inside the domain of the variable for which the function is defined. All transitions with incremental increasing and/or decreasing actions are appended with guards to prevent that the variables are updated to values outside their domains. In this paper, these guards are omitted for readability. A minimally restrictive, non-blocking, and controllable *supervisor* [5] can be synthesized from an EFA model [13]. The supervisor may be characterize as guards for the initially defined transitions [14].

The operations in Figure 2 lack explicit sequential relations. Instead, the operations are related through a set of bounded integer variables. O13 is once again used for exemplification. O13 has both a pre- and a postcondition. O13 can start if the boolean guard expression (prefixed **g:**) evaluates to true and each of the four action functions (prefixed **a:**) evaluate to a value inside the domain of the variable for which the function is defined. O13 can finish if the single action function evaluates to a valid value.

III. Explaining the concept

The sequences of operations (SOPs), that model product types, are *transformed* into an Extended Finite Automaton (EFA) model required for the approach in [10]. A supervisor is *synthesized* from the EFA model [14]. Guards are *extracted* from the supervisor [14]. Finally, in the *retransformation*, the supervisor (as a set of guards) is *applied* to the operations.

It is important so stress that any synthesis algorithm succeeded with guard extraction can be used. The focus and contribution of this paper is the transformation from the operation model to the EFA model and the retransformation of the extracted guards back to the operations.

A. The modeling approach used in [10]

The approach proposed in [10] uses two variable types to model a system in an EFA formalism: *resource variables* and *instance variables*. Each resource type, X, in the system is modeled as a resource variable, with a domain from 0 to X^c. X^c is the initial and marked value. Each product type is described as a set of *processing stages* (in the following just *stages*), where each stage requires a set of resource instances to execute. Each stage is modeled as an instance variable. The domain is from 0 to n, where n is the maximal number of instances that can execute the stage simultaneously. n can be calculated based on the available resource instances in the system. 0 is the initial and marked value. The use of instance variables gives partial order reduction benefits for the size of the model in contrast to explicit product instance models.

The *connection*, sequential relation, between two consecutive stages is modeled as a transition with unique controllable event, guards and actions. The transition is self-looped at the single location that exists in the EFA model. Three general expressions can be identified in a connection: *allocation*, *deallocation*, and *progress*. The allocation expression checks for availability of resource instances required in the target stage, and also performs the actual allocation. The deallocation expression deallocates resource instances used in the source stage. Finally, the progress expression checks for availability of an instance in the source stage, and also performs the switch between source and target instance variables.

B. A Single-Unit RAS

Example 1. The following example is adapted from Example 1 in [4]. Two product types (P1 and P2) are to be produced in a flexible automated robotic system consisting of four resource types (M1-M4), and a transporter robot, where $M1^c$=$M2^c$=$M3^c$=$M4^c$=1. Each instance of a product type is machined in three operations. The operations have sequential relations to other operations and resource requirements as given according to the product type SOPs in Figure 1. The operations are to be executed in a straight sequence. The robot is only used when both a source and a target resource are allocated, it can therefore be omitted in the deadlock calculation.

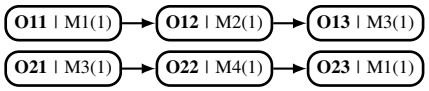

Fig. 1. The SOP for product type 1 (P1) top and for product type 2 (P2) bottom.

Transformation and retransformation are presented for the operations in P1. The operations in P2 are (re)transformed similarly but with reversed resource allocation.

1) Transformation: The transformation is based on the idea that an operation is modeled as a transition and a stage and that an operation is finished at the same time that it is started, i.e. there is no explicit state for execution. In order to include the execution state in the final result, an additional type of progress expressions are added to the operations in the retransformation.

Resources allocated for the last stage before a product exits a system, the *terminal stage*, are eventually deallocated without allocating any new resources. Deallocation will not cause deadlock. Thus, it is enough to check that the resources required for a terminal stage are available, but no allocation is necessary in the model used for deadlock analysis, see Papers IV and V in [9]. Therefore a terminal stage in [10] lack instance variable and is only modeled by a transition.

The EFA model for Example 1 contains four resource variables mk, k=1..4, and two times two instance variables oij, i,j=$1, 2$. All variables have a domain $\{0, 1\}$.

The description below shows allocation, deallocation, and progress expressions for the three transitions that model the operations in P1. A guard is prefixed with a **g:** and an action is prefixed with a **a:**. A transition can only fire if all guards and actions are satisfied [13].

o11 **a:**o11+=1 **a:**m1-=1
o12 **a:**o11-=1 **a:**o12+=1 **a:**m2-=1 **a:**m1+=1
o13 **g:**m3>=1 **a:**o12-=1 **a:**m2+=1

The progression of the product instances through the system is modeled with the progress expressions. The outside of the system is assumed never to be restrictive, so a new product instance can always enter the system if the expression for O11 is satisfied. Similarly, a product can always exit the system if the expression for O13 is satisfied. Operation O13 is modeled as a terminal stage. Deallocation of resource instances is performed in the first successor stage that does not require the instance, in this case in O12 (for M1) and in O13 (for M2).

2) Calculation of the avoidance policy: The problem to coordinate concurrent processing of P1 and P2 is reduced to synthesizing a supervisor for the EFA model [10]. The EFA model for Example 1 contains; a single location EFA with six self-loop transitions (three transitions per product type), six controllable events (one for each transition), and four variables for resources and four variables for operations. The synthesized supervisor is realized as guards over the defined EFA variables for when each controllable event can be (or can not be) executed [14], [10]. The extracted guards are not necessarily unique. Several variable value combinations can correspond to the same state in the supervisor.

3) Retransformation: In the retransformation; the supervisor is applied to the operations, explicit execution states are added to the calculated policy, and allocation and deallocation expressions are added for operations modeled as terminal stages.

The start of each operation is modeled as a transition with a unique event in the EFA model, therefore the extracted guards can be applied directly to the precondition of the corresponding operation.

In order to express that an operation has to be finished before a successor operation can start, an additional set of variables is included into the model. The new set of variables is a duplicate of the set of instance variables. An additional type of progress expressions, termed *stretched progress expressions*, are included in the model to capture that an operation progresses from executing to finished. The stretched progress expressions are used on the new variables in the same way as the progress expressions are used on the instance variables. The single difference is that the incrementation takes place in the postcondition of each operation, instead of the precondition.

The three operations in Figure 2 show allocation, deallocation, progress, stretched progress, and extracted guard for the retransformed operations in P1.

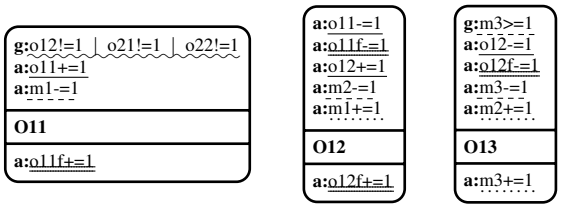

Fig. 2. Retransformed operations in P1.

Operations O11 and O21 are the single operations with extracted guards in Example 1. The guard for O11 in Figure 2 is a disjunction of instance variables.

Note that the deallocation of the M1 (M2) instance occurs in the transition that connects the two operations O11 and O12 (O12 and O13) in the EFA model. Thus, at least one M1 (M2) instance is available in the target state for these transitions in the synthesis. The introduction of the stretched progress expressions will divide all these target states into execution and finish states, so the resource instance should eventually be deallocated in one of these states. Therefore, the deallocation of the M1 (M2) instance in operation O12 (O13) can be an action in the pre- or the postcondition and still comply with the supervisor. In Figure 2, the deallocation is a preaction. This discussion is continued in Section V.

C. To summarize

This section has shown how operations that model machining of two product types can be adapted to operations that control concurrent machining of multiple product instances without resource deadlocks and with maximal flexibility. The initial conditions that model sequential relations and resource requirements are replaced with conditions that use

the resource variables and two times the instance variables defined in the approach in [10].

For the discussion to follow, the operations in P1 are classified as *start* (O11), *sequence* (O12), and *terminal* operation (O13), respectively.

IV. Complex machining and assembly

Many systems are more flexible than the system in Example 1. A product can be machined in alternative branches and/or by alternative resources. New product instances can originate from sub-instances. Therefore, this section describes transformation and retransformation for SOPs with different types of alternatives and assembly.

A. A Conjunctive/Disjunctive RAS with assembly

Example 2. A system is to perform machining on sub-products (P5, P6, and P7) and thereafter assemble the sub-products into products (P8 and P9). The SOPs for the product types are given in Figures 3 and 5-7. The system consists of five resource types: $M9^c$=1, $M5^c$=$M6^c$=$M8^c$=2, and $M7^c$=3. The system allows the (sub-) products to be machined and assembled in alternative branches. Two types of alternatives are identified: *controllable* and *uncontrollable* alternatives. In a controllable alternative, all branches are equally good from a product point of view, therefore any branch can be chosen. In an uncontrollable alternative, the branch is chosen online based on the outcome from a predecessor operation.

The resource requirements in the operations for P5 (P6 and P7) are conjuncted with one instance of a movable fixture of type Fa (Fb), Fa^c=2 and Fb^c=3. Allocation of more than one resource type at the same time is an example of a Conjunctive RAS [4].

Transport of products between resources and transport of empty fixtures are only performed when both source and target resources are allocated. Transport can therefore be omitted in the deadlock analysis.

1) A controllable alternative: The SOP for P5, see Figure 3, contains a controllable alternative, the system can choose left or right branch for each product instance. This is an example of a Disjunctive RAS [4].

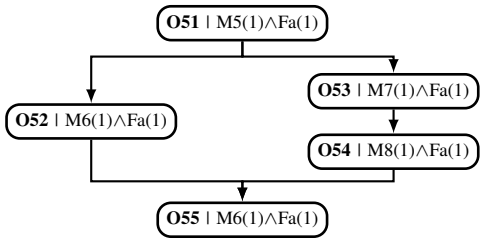

Fig. 3. SOP for machining of sub-product P5.

All operations for P5 except O55 are transformed as start or sequence operations. Each branch in the alternative may require different allocation and deallocation expressions and for sure different progress expressions when they merge. A stage which follows directly after the merge of alternative branches is therefore transformed into as many transitions as there are alternative branches [10]. A unique event is added for each transition. Thus, operation O55 is transformed into two transitions, in the following termed o55l and o55r. Each instance of P5 will eventually be assembled, O55 is therefore modeled with one instance variable in the EFA model.

Resource allocation and deallocation will not take place between two consecutive operations that have the same resource requirements. The fixture instance allocated in operation O51 is therefore used in all operations O5x.

The description below shows the transitions for P5 in the EFA model for Example 2.

o51	**a:**o51+=1	**a:**m5-=1	**a:**fa-=1	
o52	**a:**o51-=1	**a:**o52+=1	**a:**m6-=1	**a:**m5+=1
o53	**a:**o51-=1	**a:**o53+=1	**a:**m7-=1	**a:**m5+=1
o54	**a:**o53-=1	**a:**o54+=1	**a:**m8-=1	**a:**m7+=1
o55l	**a:**o52-=1	**a:**o55+=1		
o55r	**a:**o54-=1	**a:**o55+=1	**a:**m6-=1	**a:**m8+=1

All operations for P5 except O55 are retransformed as start or sequence operations. Operations that are transformed into multiple transitions, where each transition allocates and deallocates different resources, are retransformed into the same number of operation instances in the final result. The use of a unique event for each transition enables the guards extracted from the supervisor to be applied directly to each operation instance. It is important to stress that each operation instance is executed in the same way, it is only the preconditions that are different. Figure 4 shows operation O55 after retransformation. Each product instance of P5 executes O55l or O55r. The machining of a product instance is the same in both O55l and O55r. Note that only O55r is applied with an extracted guard.

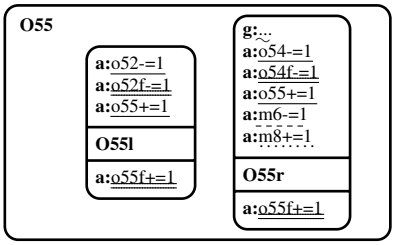

Fig. 4. Retransformation of operation O55. The guard expression in O55r is left out for clarity.

2) An uncontrollable alternative: The SOP for P6, see Figure 5, contains an uncontrollable alternative. The outcome of operation O62 determines if the left or the right branch should be executed for each product instance. This can for example be used to model abnormal behavior as in [10].

The deadlock avoidance policy must assure that an instance of P6 can execute both operation O63 followed by O64, and O65 followed by O66, when the product instance is finished with O62. This is not the case in a controllable alternative, as in the SOP for P5, where it is enough that an instance can execute one of the branches in the alternative. The approach in [10] captures this online choice with *uncontrollable events*, events that cannot be disabled by the supervisor [5]. The first operations in an uncontrollable alternative are transformed into two transitions and two instance

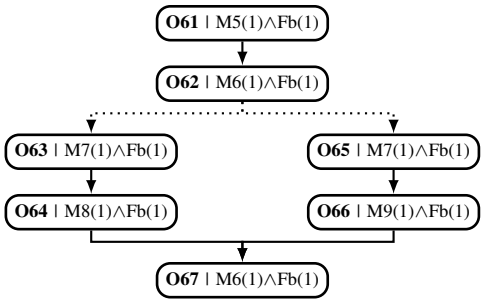

Fig. 5. SOP for machining of sub-product P6. The dotted line, to visualize an uncontrollable alternative, is an extension to the graphical language Sequences of Operations [1].

variables in the EFA model. A unique uncontrollable event is applied to the first transition and a unique controllable event is applied to the second transition. Progress expressions are added to the transitions in order to progress to the first instance variable with the uncontrollable transition and then progress to the second instance variable with the controllable transition. Resource allocation and deallocation expressions are added to the controllable transition. The first instance variable models that a product is still in the predecessor operation, but the outcome of which branch to choose has been determined. The second instance variable models that the product is in the first operation of the chosen branch.

The description below shows the transitions connected to the uncontrollable alternative for P6 in the EFA model for Example 2.

o63uc **a:**o62-=1 **a:**o63c+=1
o63 **a:**o63c-=1 **a:**o63+=1 **a:**m7-=1 **a:**m6+=1
o64 **a:**o63-=1 **a:**o64+=1 **a:**m8-=1 **a:**m7+=1
o65uc **a:**o62-=1 **a:**o65c+=1
o65 **a:**o65c-=1 **a:**o65+=1 **a:**m7-=1 **a:**m6+=1

The retransformation of the operations in P6 is similar to the retransformation as described for the operations in P5.

The uncontrollable transitions are only included to guarantee that each branch can be chosen in the synthesis. These transitions are therefore removed in the retransformation. This removal requires modification of the progress expressions for the first operations in the uncontrollable alternative.

The first operation that follows directly after the merge of uncontrollable alternative branches is retransformed in the same way as if it was a controllable alternative.

3) Alternatives in sequence: The SOP for P7 is given in Figure 6. The disjunctive resource requirements in operations O73 and O74 are syntactic sugar for a controllable alternative with a single operation in each branch. Operation O72 requires two instances of resource M7 in order to execute. The upper limit for the instance variable for O72 is given through truncated integer division as $min\{\mathrm{M7}^c/2,\ \mathrm{Fb}^c\}$.

Alternatives in sequence raise the number of transitions in the EFA model in order to capture the different allocation and deallocation combinations. The description below shows some of the transitions for P7 in the EFA model for Example 2.

o72 **a:**o71-=1 **a:**o72+=1 **a:**m7-=2 **a:**m5+=1
o73 **a:**o72-=1 **a:**o73+=1 **a:**m7+=1

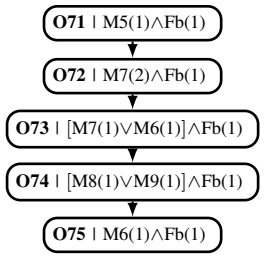

Fig. 6. SOP for machining of sub-product P7. The disjunctive resource requirement is short for a controllable alternative.

o73' **a:**o72-=1 **a:**o73'+=1 **a:**m6-=1 **a:**m7+=2
o74l **a:**o73-=1 **a:**o74+=1 **a:**m8-=1 **a:**m7+=1
o74r **a:**o73'-=1 **a:**o74+=1 **a:**m8-=1 **a:**m6+=1
o74'l **a:**o73-=1 **a:**o74'+=1 **a:**m9-=1 **a:**m7+=1
o74'r **a:**o73'-=1 **a:**o74'+=1 **a:**m9-=1 **a:**m6+=1

The retransformation of the operations in P7 follows the retransformation for the operations in P5.

4) Assembly: The SOPs for P8 and P9 are given in Figure 7. Each P8 (P9) instance is assembled from one P5 and one P6 instance (one P5 and one P7 instance). This is graphically specified with operations O55 and O67 (O55 and O75) in parallel before O81 (O91) can start.

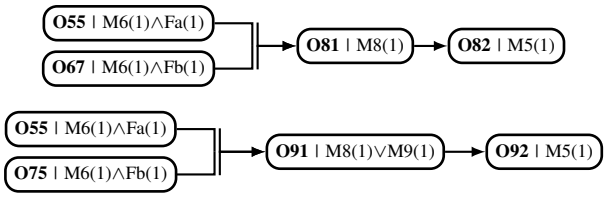

Fig. 7. The SOP for P8 top and P9 bottom.

Transformation for assembly situations are not included in the approach in [10]. A straightforward extension, to include assembly in the EFA model, is to let the progress expression for an assembly operation require a product instance in each of the ingoing branches. The deallocation expression deallocates all resource instances that are not used in the assembly operation.

The description below shows the transitions for P9 in the EFA model for Example 2. Operations O82 and O92 are transformed as terminal operations.

o91 **a:**o55-=1 **a:**o75-=1 **a:**o91+=1 **a:**m8-=1 **a:**m6+=2 **a:**fa+=1 **a:**fb+=1
o91' **a:**o55-=1 **a:**o75-=1 **a:**o91'+=1 **a:**m9-=1 **a:**m6+=2 **a:**fa+=1 **a:**fb+=1
o92l **g:**m5>=1 **a:**o91-=1 **a:**m8+=1
o92r **g:**m5>=1 **a:**o91'-=1 **a:**m9+=1

Operations O81, O91, and O91' are retransformed as sequence operations. O82 and O92 are retransformed as terminal operations. Note that O92 is retransformed like O55, because of the alternative in O91.

5) Computational results: The EFA model for Example 2 has 75135 reachable states. The supervisor for the model contains 68118 states. 19 out of the 32 events in the EFA model are applied with extracted guards. The extracted guards are too complex to be understood by a human.

Transformation, synthesis ([14]), guard extraction ([14]), and retransformation take a few seconds on a standard computer.

V. Deadlock arising from incorrect deallocation

As indicated in Section III-B.3, in order to comply with the supervisor, resource instances should be deallocated in the first successor operation to an operation where the resource instances are not required. It is rather straightforward to see that deallocation in another successor operation than the first successor operation can cause deadlock. The synthesized control policy is not valid.

A slightly modified Example 1, see Figure 8, is included as observation. The M1 instance used in operation O11 is deallocated in O13 instead of O12, as required. State `p` is then a deadlock state because the modified deallocation gives no M1 instance available in state `p`. A correct deallocation would have enabled operations O11 and O23 to start from state `p`. Both O11 and O23 require a M1 instance.

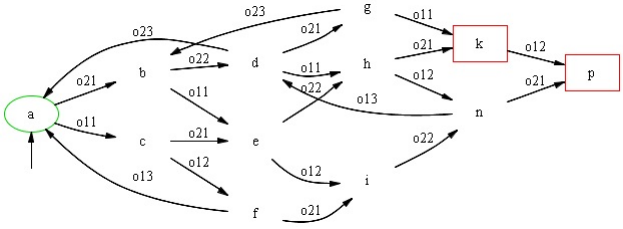

Fig. 8. Synchronization of the retransformed operations from the modified Example 1. Only start events are included for clarity.

It is maybe less obvious to see that deallocation of a resource instance as an action in the postcondition to the operation where it has been used can also cause deadlock. An example is given in Figure 9. The SOPs for P3 and P4 are given in Figure 10. The system has the same resources as in Example 1. The M1 instance used in O31 is deallocated with an action in the postcondition to O31 and not in O32, as is required in the approach. State `e` is then a deadlock state because neither O32 nor O42 can start. O41 is not enabled in state `d` if the deallocation would have been performed in O32.

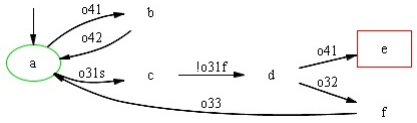

Fig. 9. Synchronization of the retransformed operations given in Figure 10, with incorrect deallocation. Only start events are included for clarity, except for O31.

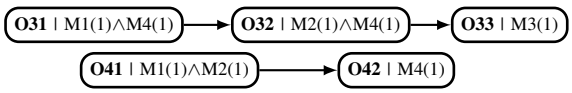

Fig. 10. The SOPs for P3 top and P4 bottom.

VI. Conclusion

This paper has shown how to adapt initial defined sequences of operations that model machining and assembly of single product instances of different types to operations that control concurrent manufacturing of multiple product instances. The proposed approach can handle initially defined sequences of operations with routing flexibility. Two types of flexible routing are described, controllable and uncontrollable alternatives. An operation may have a conjunctive and/or disjunctive resource requirement. The concurrent manufacturing is without resource deadlocks and with maximal flexibility. The formal deadlock avoidance approach presented in [10] is used as a back-end.

As outlook and future work, the approach in this paper can be refined and improved from the following aspects:

- In order to increase the number of systems for which the approach is applicable, study how to model resources that cannot be abstracted with a binary variable.
- The approach in [10] suffers from state space explosion for large system sizes, it is therefore interesting to find variable encodings that can mitigate this problem.

References

[1] B. Lennartson, K. Bengtsson, C. Yuan, K. Andersson, M. Fabian, P. Falkman, and K. Åkesson, "Sequence Planning for Integrated Product, Process and Automation Design," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 4, pp. 791–802, 2010.

[2] E. G. Coffman, M. J. Elphick, and A. Shoshani, "System Deadlocks," *Computing Surveys*, vol. 3, no. 2, pp. 67–78, 1971.

[3] K. Bengtsson, *Operation Specification for Sequence Planning and Automation Design*. Licentiate Thesis, Chalmers University of Technology, 2009.

[4] S. A. Reveliotis, "Resource allocation systems: concepts and problems," in *Real-Time Management of Resource Allocations Systems A Discrete Event Systems Approach*, ch. 1, Springer, 2005.

[5] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[6] E. M. Gold, "Deadlock Prediction: Easy and Difficult Cases," *SIAM Journal on Computing*, vol. 7, no. 3, pp. 320–336, 1978.

[7] J. Park and S. A. Reveliotis, "Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings," *IEEE Transactions on Automatic Control*, vol. 46, no. 10, pp. 1572–1583, 2001.

[8] Z. Li, M. Zhou, and M. Jeng, "A maximally permissive deadlock prevention policy for FMS based on Petri net siphon control and the theory of regions," *IEEE Transactions on Automation Science and Engineering*, vol. 5, no. 1, pp. 182–188, 2008.

[9] K. Åkesson, *Methods and Tools in Supervisory Control Theory - Operator aspects, computational efficiency, and applications*. PhD Thesis, Chalmers University of Technology, 2002.

[10] Z. Fei, S. Miremadi, and K. Åkesson, "Modeling sequential resource allocation systems using Extended Finite Automata," in *IEEE International Conference on Automation Science and Engineering*, pp. 444–449, 2011.

[11] E. Roszkowska, "Supervisory Control for Deadlock Avoidance in Compound Processes," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 34, no. 1, pp. 52–64, 2004.

[12] M. Fanti, G. Maione, and B. Turchiano, "Design of supervisors to avoid deadlock in flexible assembly systems," *International Journal of Flexible Manufacturing Systems*, vol. 14, no. 2, pp. 153–171, 2002.

[13] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of Discrete Event Systems using Finite Automata With Variables," in *IEEE Conference on Decision and Control*, pp. 3387–3392, 2007.

[14] S. Miremadi, K. Åkesson, and B. Lennartson, "Symbolic Computation of Reduced Guards in Supervisory Control," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754–765, 2011.