Derivation of placement transitions for offline calculation of restart states

Derivation of placement transitions for offline calculation of restart states

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

Abstract

This paper presents a preprocess to an existing method for offline calculation of restart states for manufacturing systems modeled by operations. In the existing method, placement transitions are used to model restart in restart states from potential error states, and supervisory control theory is used to calculate which of these transitions are valid. With the proposed preprocess, the precedence and the alternative dependencies between the operations are exploited in order to reduce the number of such placement transitions which are required in the model used by the existing method. With such a reduced model, the valid restart states for larger and more complex systems can be calculated.

1 Introduction

Error recovery in manufacturing systems is a complicated task [2], often divided into three major activities [10]: *detection* of discrepancies between the intended behavior and the actual behavior of the system, *diagnosis* to find the original fault causing the observed error, and *recovery* of the system to continue the nominal production. Recovery is often partitioned into *error correction* to remove underlying faults and *restart* to resynchronize the control function in the control system and the resources such that the nominal production can be resumed [12].

The control function for a manufacturing system is often modeled by a set of *operations* that are to be executed in order to refine a product [3, 9]. Possible operation sequences emerge through *dependencies* between the operations. These dependencies may arise both due to product as well as manufacturing requirements [3].

When restarting a manufacturing system, consideration has to be made for the possible *reexecution requirements* on the operations [4]. An operation can be non-reexecutable or its reexecution ability may depend on the progress of other operations [1, 11].

Each operation is typically modeled by three states; an *executing state* is preceded and succeeded by states to model that the operation has not started and is completed, respectively. The control function can then be seen as a set of *control function states*, where each state models that some operations have not started, some operations are executing, and that the rest of the operations are completed. Moreover, a *physical state* of the manufacturing system may be seen as a composition of internal resource states. Since resources are used when the operations are executed, several physical states *correspond* to each control function state.

An error in the system can now be seen as a physical state that does not correspond to the current control function state. This control function state is often referred to as the *error state* [11, 21].

The objective in most error recovery methods, presented in the literature, is then to restart the system such that the nominal production can continue from an earlier, later, or the current control function state with respect to the error state. Recovery in an earlier or a later state is often referred to as *backward* and *forward error recovery*, respectively. The state from which the control function is restarted is often called *restart state* [1, 4] or restart point [12].

The restart methods presented in the literature are either *online* methods, such as [7, 19–21], or *offline* methods, such as [1, 4, 5, 12, 16]. For online methods, one or more restart states are calculated at the time of the error in contrast to offline methods where the restart states are precalculated. Typically, online methods require heavier calculations to be performed during the actual restart than offline methods. Offline methods, on the other hand, enable evaluation of the calculated restart states beforehand, which allows much more elaborate calculations than can typically be performed on industrial computing hardware on the shop-floor.

Moreover, the beforehand calculation enables different restart alternatives to be analyzed already when the production in the manufacturing system is planned, such that undesirable situations can be resolved if possible. The result of the analysis can then be (automatically) refined into comprehensible operator instructions to use during the online restart phase. This beforehand analysis is a big advantage for the offline methods.

With the restart states precalculated, the online restart phase of an offline method, can be reduced to four

*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.

straightforward steps. First, the operator selects a restart state from the precalculated restart states. The restart states can for example be stored in a data base connected to the control system. Second, the active state of the control system is update to the selected restart state. It is thus assumed that a mechanism for state transitions is available in the control function. Third, the operator places the manufacturing system in a physical state that corresponds to the selected restart state. The operator is beneficially guided by instructions for how to reach this physical state. Finally, the nominal production can be (re)started by the operator.

In earlier work [4], an offline method for calculation of restart states is proposed. By aiming at backward error recovery, it is possible to compensate for unsatisfactory refinement as a consequence of an error and keep extensions to the control function at a minimum.

In the method, *placement transitions* are used to model restart of the control function in restart states from potential error states. Thus, a graph based model can be built up from a given set of operations, their dependencies, possible reexecution requirements, and the full set of placement transitions. This model is referred to as the *restart model*. This full set of placement transitions contains all placement transitions that are required in order to connect each potential error state with all of its restart states.

The name *placement* alludes to the operator action to *place* the manufacturing system in a physical state as part of the online restart phase. A placement transition accomplish a similar action, but in the control function.

Due to the dependencies and the reexecution requirements, not all of the placement transitions model restart in *valid restart states*. A valid restart state is a control function state from which the nominal production can continue and eventually complete such that all dependencies and reexecution requirements are always satisfied. Therefore, a supervisor [15] is synthesized for the restart model. The valid restart states will then correspond to the target states for the placement transitions which are enabled by the supervisor.

To guarantee that all valid restart states are calculated for a set of operations, the full set of placement transitions for the set of operations can in theory be included in the restart model. The synthesis algorithm assures that none of the transitions that violate any dependency or reexecution requirement are enabled by the supervisor.

From a practical point of view, the problem is, however, that this full set of placement transitions grows exponentially with the number of operations that are included in the restart model. Therefore, the number of placement transitions to include in the restart model has to be significantly lower to cope with systems of industrial sizes. Thus, to enable error recovery for industrial systems it is critical to enhance the method to increase the upper limit for the number of operations that can be included.

Therefore in this paper, the precedence and the alternative dependencies between the operations are exploited in order to decrease the number of placement transitions in the restart model, still guaranteeing that all valid restart states are calculated in the succeeding synthesis. The idea is that the placement transitions to use in the restart model are preprocessed, based on a compact model, such that only the transitions that satisfy the precedence and alternative dependencies are included. This preprocessing will increase the upper limit for the number of operations that can be included in the restart model such that all valid restart states can be derived.

This paper is organized as follows: in Section 2 background is given to the formal models and methods used and to the restart method proposed in [4], respectively. An overview of the main idea for this paper is given in Section 3 and presented in detail in Section 4. Finally, in Section 5 quantitative studies on three manufacturing systems are performed and in Section 6 conclusions and suggestions for future work are provided.

2 Preliminaries

This section presents the premises for this paper. First the modeling formalism is presented. Thereafter, this formalism is used to model the control function in a control system for a manufacturing system. Finally, it is described how restart states are calculated for such a model of the control function according to the method presented in [4].

2.1 Automata

A *deterministic finite automaton*, in the following just automaton, is a 4-tuple $A := \langle Q_A, \Sigma_A, \delta_A, q_A^0 \rangle$ where Q_A is the non-empty finite set of *states*; Σ_A is the non-empty finite set of *events*, the *alphabet* for the automaton; $\delta_A : Q_A \times \Sigma_A \to Q_A$ is the partial transition function and $q_A^0 \in Q_A$ is the *initial state*.

Let $\delta_A(q, e)!$ denote that an event e is *defined* from a state q in an automaton A. A transition $\langle q, e, p \rangle \in \delta_A$ is said to be *enabled* from the state q. When the transition is *fired* the active state of the automaton A is *updated* to the *target state* p. The *active event function* returns the set of events defined from q in A, $\Gamma_A(q) := \{e \in \Sigma_A | \delta_A(q, e)!\}$.

The set of all finite sequences of events over an alphabet Σ_A including the empty sequence, ε, is denoted Σ_A^*. An element $s \in \Sigma_A^*$ is called a *string*. The *suffixes* to a string s are given as $\{u | s = tu, t \in \Sigma_A^*, u \in \Sigma_A^*\}$.

A *language*, denoted $L(A)$, is the set of strings generated by the automaton A from the initial state q_A^0. The *suffix closure of a language* $L(A)$ is the set of all suffixes to all strings in the language $L(A)$.

Interaction of two automata may be modeled by full synchronous composition [6]. The *full synchronous composition* (FSC) of two automata A and B is defined as $C := A||B$ where $Q_C := Q_A \times Q_B$; $\Sigma_C := \Sigma_A \cup \Sigma_B$; $q_C^0 := \langle q_A^0, q_B^0 \rangle$ and $\delta_C\big(\langle q_A, q_B \rangle, e\big) :=$

$$\begin{cases} \langle \delta_A(q_A, e), \delta_B(q_B, e) \rangle & e \in \Gamma_A(q_A) \cap \Gamma_B(q_B) \\ \langle \delta_A(q_A, e), q_B \rangle & e \in \Gamma_A(q_A) \setminus \Sigma_B \\ \langle q_A, \delta_B(q_B, e) \rangle & e \in \Gamma_B(q_B) \setminus \Sigma_A \\ undefined & otherwise \end{cases}$$

2.2 Modeling the control function with operations

In this paper, a *manufacturing system* contains a set of *resources*. The production is supervised by a *control system* that executes a *control function*. The processes and tasks that are to be executed by the control function are modeled by *operations*. The set of operations modeled for a system is denoted Ω and the number of operations in this set as $|\Omega|$. An operation requires a subset of the resources in order to execute.

The basic assumption is that all operations are running in parallel. To model the *nominal production*, this parallel execution between the operations can be restricted by *dependencies*. Example of such dependencies are *precedence*, *alternative*, and *arbitrary order*.

A *straight operation sequence*, in the following just operation sequence, can be split up into its *subsequences*. For three operations A, B, and C one possible operation sequence is ABC with non-empty subsequences: ABC, AB, BC, A, B, and C. The last operation in any non-empty operation (sub)sequence is called *last operation* whilst the the set of preceding operations are called *init operations*. Thus, a sequence with a single operation has no init operations.

An operation $k \in \Omega$ may formally be modeled by an automaton A_k with three states and two unique events [3, 9], see Figure 1. The three states, denoted i, e, and c, model that the operation is *initial* (not started), *executing*, and *completed*, respectively. The *start event*, denoted $\uparrow_k$, labels the transition from the initial state to the executing state and similarly the *complete event*, denoted $\downarrow_k$, labels the transition from the executing state to the completed state.

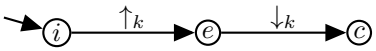

Figure 1. Automaton model of an operation k.

Given the automaton for a single operation, the control function may be modeled by FSC of all operations in Ω denoted A_Ω, where $A_\Omega := ||_{k \in \Omega} A_k$. Each state in A_Ω is therefore a combination of local operation states. Thus, in each state $q \in Q_{A_\Omega}$, the operations in Ω can be partitioned into three disjoint sets, where the operations in each set are in their initial, executing, and completed states, respectively. The distribution of operations between the different sets may be seen as a measurement of the overall *progress* of the production.

To be able to relate this progress for any two states in Q_{A_Ω}, the concept of upstream is introduced. For two states $p, q \in Q_{A_\Omega}$, q is *upstream* of p if there exists a non-empty set of non-initial operations in p, denoted $\mathcal{O}$, such that all these operations are initial in q and the remaining operations, $\Omega \setminus \mathcal{O}$, keep the same local operation state in q as in p. Thus, all states in A_Ω have at least one upstream state except the initial state $q^0_{A_\Omega}$, where all operations are initial.

An operation is *reset* when its active state is updated from a non-initial state to the initial state. Note that, additional transitions have to be included in the operation model in order to reset the operation. Such transitions will be introduced in the next subsection.

When restarting a system, there can be additional constraints, other than the dependencies, between the operations that have to be satisfied, so called *reexecution requirements* [4]. An operation can for example be non-reexecutable or its reexecution ability can be based on the progress of the production [1, 11].

2.3 Calculation of restart states

In [4] a method for calculation of restart states is proposed. The method is based on supervisory control theory [15]. For a set of operations, their dependencies, and reexecution requirements, it is shown how restart states can be derived through synthesis of a supervisor.

In the method, it is assumed that an error can occur when any of the resources in the manufacturing system are used. That is, at least one operation is executing. Thus, a state in the control function is a *potential error state* if at least one of the operations is in its executing state.

To compensate for unsatisfactory refinement as a consequence of an error, the method aims for backward error recovery. To have a well defined notion of backward, a restart state is always upstream of an error state.

An upstream state is a *valid restart state* if the nominal production can be restarted from this state and eventually complete, such that none of the dependencies nor any of the reexecution requirements are violated.

Given these prerequisites, the overall aim of the method is then to derive *all* valid restart states for *all* potential error states. With such an open approach, different postprocesses can be applied on the result to meet different desires. As two examples, Andersson et. al. aim to minimize the set of valid restart states with the constraint that all potential error states have at least one valid restart state [2]. For other systems it can be desirable to select one valid restart state for each potential error state, such that the operator action to place the manufacturing system in a corresponding physical state becomes as simple as possible, according to some appropriate metric.

In the method, the derivation of valid restart state is accomplished through synthesis of a supervisor for a set of automata that model the operations, their dependencies, reexecution requirements, and restart in all upstream states for all potential error states. Each operation is modeled by an automaton as given in Figure 1. The dependencies and reexecution requirements are modeled by automata [4], the details are outside the scope of this pa-

per. The restart is modeled by placement transitions, as described below. These automata together are termed the *restart model*.

A *placement transition* models restart in an upstream restart state from a set of potential error states. Each transition models reset of one *error operation* k and a set of *reset operations* $\mathcal{O}$, labeled by the unique controllable event $\sigma_{k:\mathcal{O}}$. Note that $\mathcal{O}$ can be the empty set. The transition is enabled from all states in the synchronous product of the restart model where k is executing and each of the operations in $\mathcal{O}$ is executing or completed. The target state for the transition is the upstream state in the synchronous product where k and all operations in $\mathcal{O}$ are initial.

This modeling of placement transitions is illustrated by a simple example with two operations B and D. Without dependencies and reexecution requirements, the synchronous product of the corresponding restart model is as shown in Figure 2. For clarity of presentation, only the placement transitions for D are considered. Since there are two operations in the example system, there will be two unique events with D as error operation, σ_D and $\sigma_{D:B}$. The event σ_D models reset of just D and $\sigma_{D:B}$ models reset of both D and B. Note the simplification in the indexes, when $\mathcal{O}$ is the empty set it is not written out, and for non-empty $\mathcal{O}$ the operations are simply written as a sequence.

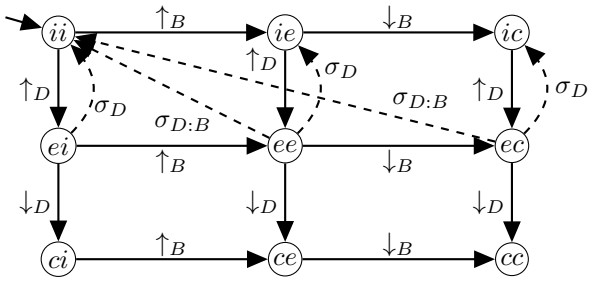

Figure 2. Parallel execution of the two operations B and D. Placement transitions for D are dashed.

Operation D executes in the three states in the middle row of the automaton in Figure 2. Reset of just D is enabled in all three executing states, the placement transitions labeled by σ_D. Reset of both D and B is only enabled when B has started. Thus, the placement transitions labeled by $\sigma_{D:B}$ are enabled from the two rightmost executing states for D.

Due to the dependencies and the reexecution requirements, not all of the placement transitions model restart in valid restart states. Thus, a supervisor [15] is synthesized for the restart model. The valid restart states will then correspond to the target states for the placement transitions that are enabled by the supervisor. Note that the method formulates a general synthesis problem that is independent of the solver. Thus, efficient algorithms such as compositional synthesis [14] and/or symbolic synthesis [13] can be used.

Finally, in the worst case all operations are executing in parallel. A state in the synchronous product of the restart model that contains an executing state for an operation will then have up to $2^{|\Omega|-1}$ upstream states. Since there are $|\Omega|$ operations, the number of transitions in the *full set of placement transitions* for the restart model is then given as $|\Omega| \times 2^{|\Omega|-1}$.

Due to the modular modeling of the operations, the same placement transition will occur in multiple transition functions. In the above calculation, each placement transitions is, however, only calculated once. From Figure 2, the placement transition where D and B are error and reset operations, respectively, will for example occur as three transitions in the two transition functions δ_{A_D} and δ_{A_B}, such that $\langle e, \sigma_{D:B}, i\rangle \in \delta_{A_D}$, $\langle e, \sigma_{D:B}, i\rangle \in \delta_{A_B}$, and $\langle c, \sigma_{D:B}, i\rangle \in \delta_{A_B}$.

3 The overall idea

As outlined in Section 2, in [4] the full set of placement transitions is included in the restart model. By applying synthesis on the restart model, the valid restart states are derived through the target states of the enabled placement transitions.

The major drawback with the method is the exponential growth of placement transitions for each additional operation that is included in the restart model. This disadvantage makes the method intractable for large systems. Already for a moderate system where $|\Omega| = 20$ the number of placement transitions is in the order of 10^7.

To overcome this problem, this paper suggests that the set of placement transitions to include in the restart model is correlated with the dependencies. Only the placement transitions that are *valid with respect to the precedence and the alternative dependencies* are to be included in the restart model. A placement transition is valid with respect to these two types of dependencies if the nominal production can be restarted in the restart state that the transition models and eventually complete, such that none of the precedence nor any of the alternative dependencies are violated.

With fewer placement transitions in the restart model, the upper limit for the number of operations to include in the calculation can be increased. Thus, systems of industrial sizes can be handled.

In this paper, these valid placement transitions are identified through traversing a more compact model than the restart model itself. Thus, the identification can be performed more efficiently than in the synthesis.

It is important to stress that, the presented method is a preprocess to the synthesis of the restart model as described in [4] and not a substitute. The succeeding synthesis guarantees that none of the placement transitions that violate other types of dependencies than precedence and alternative and any of the reexecution requirements are enabled. Note however that in the special case with

only precedence and alternative dependencies and no re-execution requirements between the operations, this pre-process is enough for derivation of the valid restart states and the synthesis can be skipped.

The next section shows how to construct the compact model and how to derive the valid transitions from traversing this model.

4 Valid placement transitions

This section shows how to derive the placement transitions that are valid with respect to the precedence and the alternative dependencies. These valid transitions are derived from a compact model of all the topological orderings of operations that are possible with respect to the dependencies. The derivation is first performed on a model only with precedence dependencies. Thereafter, the alternative dependencies are included in the model.

4.1 Precedence dependencies

A first approach to derive the set of *precedence-valid placement transitions* is to start with the full set of placement transitions and simply remove those that do not satisfy the precedence dependencies. The approach is illustrated by an example system with three operations: A, B, and C, and two precedence dependencies: A precedes B and B precedes C.

The placement transitions where A is the error operation and B is among the reset operations are not precedence-valid and can be removed. From the definition of placement transitions, these transitions are enabled from states where A is executing and B is executing or completed. From the dependency that A precedes B, A has to complete before B can execute. So following the dependency, the states where the transitions are enabled will never be reached. Thus, the transitions are not precedence-valid and can therefore be removed. Similarly, the placement transitions where B is the error operation and C is among the reset operations are not precedence-valid.

From the two dependencies, it is quite straightforward to see that A will precede C. Removal of additional placement transitions due to such transitive dependencies, indirect relations, is however not covered in this first approach. Thus, more non-precedence-valid placement transitions can be identified and removed if these transitive dependencies are exploited.

Given the precedence dependencies, to fully exploit these transitive dependencies all topological orderings, sorting arrangements, [8] of the operations can be studied. In this context, a topological ordering is an ordering of the operations, such that if the operations are executed sequentially all precedence dependencies are satisfied. Thus, a placement transition is then precedence-valid if and only if there is an ordering where all of the reset operations precede the error operation.

Thus, the second approach to derive the set of precedence-valid placement transitions is then to identify all topological orderings and thereafter derive the transitions from these orderings. Once again from the definition of placement transitions, a placement transition is enabled when its corresponding error operation is executing and each of the reset operations are either executing or completed. Since each ordering is an operation sequence, the ordering can be split up into its non-empty set of subsequences. From the requirement of sequential execution, all init operations in a subsequence are completed before the last operation in the subsequence can execute. This last operation and these init operations will then correspond to the error operation and the reset operations in a precedence-valid placement transition.

Some remarks, the subsequences with a single operation correspond to placement transitions without reset operations, these transitions will model reset of a single (error) operation. No precedence-valid placement transitions are missed, due to the requirement of sequential execution. Since all topological orderings are evaluated, all orderings of any parallel operations are eventually considered. Moreover, since the ordering of the reset operations is unimportant, some subsequences will correspond to the same placement transition.

After this initial motivation for how the set of precedence-valid placement transitions can be derived, the remainder of this subsection will focus on modeling the operations and the dependencies in order to derive this set in practice. The example with the operations A to C is also revised.

The union of all the subsequences from all topological orderings can be derived through FSC of a set of automata. By modeling each operation as a two-state automaton with a unique event labeling the single transition between the two states and each precedence dependency as an automaton for a constrained event sequence, the union of the subsequences corresponds to the strings in the suffix closed language generated by the synchronous product for these automata. From the definition of FSC, the synchronized product will contain all strings that are allowed for the automata such that the shared events are executed simultaneously. In the following, this synchronous product for derivation of the topological orderings is termed the *dependency automaton*.

For the three operations A, B, and C, there is only a single topological ordering: ABC. To enable a more interesting discussion, the example is extended with an additional operation D and the extra precedence dependency that A precedes D. The addition gives three possible orderings in the example system: $ADBC$, $ABDC$, and $ABCD$.

The operations A to D are modeled by seven automata, see Figure 3. For simplicity, let the unique event for each operation be denoted by the operation name itself. In the left generic automaton the label I will then take the values $I \in \{A, B, C, D\}$. The two rightmost automata

model the three precedence dependencies, where the label J takes the values $J \in \{B, D\}$. The *dependency automaton* is shown in Figure 4.

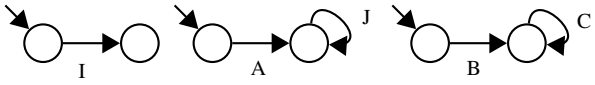

Figure 3. To the left, a generic automaton for the operations A, B, C, and D. The two rightmost automata model the three precedence dependencies. The labels I and J take the values $I \in \{A, B, C, D\}$ and $J \in \{B, D\}$.

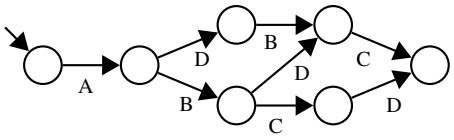

Figure 4. The dependency automaton for the four operations A, B, C, and D, with respect to precedence dependencies.

The suffix closed language generated by the dependency automaton for the example operations, given in Figure 4, contains 20 unique strings. Four of the strings with C as the last event/operation demonstrate that multiple strings/subsequences may correspond to the same placement transitions. Both of the two strings BDC and DBC will for example correspond to the placement transition where C is the error operation and B and D are the reset operations. From the 20 strings, 18 precedence-valid placement transitions can be derived. Thus, $4 \times 2^{4-1} - 18 = 14$ placement transitions can be removed from the restart model, still guaranteeing that all valid restart states can be calculated.

4.2 Alternative dependencies

For any alternative in the nominal production, only the operation(s) in one of the branches will be executed. Placement transitions that reset operations from different branches in an alternative are then only enabled from states which are unreachable from the initial state in the supervisor for the restart model. Thus, these transitions will not be enabled in the supervisor. As for the precedence dependencies, the alternative dependencies can be exploited such that only the *alternative-valid placement transitions* are included in the restart model.

It is straightforward to model these alternative dependencies by automata and thereafter include these automata in the approach described for precedence dependencies. The suffix closed language generated by the synchronized dependency automaton will then contain strings that correspond to the placement transitions that are both precedence-valid and alternative-valid.

An alternative dependency between any two operations can be modeled by an automaton with two states. The two states are connected with two transitions labeled by the events that model the two operations. One event is labeled on each transition. From the definition of FSC, only a single event/operation can be fired.

For demonstration, an alternative dependency is added between the operations C and D in the example with the four operations. The alternative is modeled by the automaton to the left in Figure 5. The dependency automaton is shown to the right in Figure 5. As expected, there are no strings that contain both C and D. The dependency automaton contains twelve strings, that is, eight fewer than for the dependency automaton without the automaton for the alternative dependency between C and D. As before, the valid placement transitions are derived through identification of all unique strings in the suffix closed language generated by the dependency automaton. Twelve both precedence-valid and alternative-valid placement transitions are derived, which means that in total $4 \times 2^{4-1} - 12 = 20$ placement transitions can be removed.

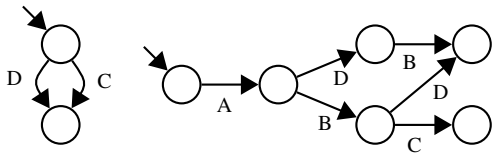

Figure 5. The left automaton models alternative dependency between operations C and D. To the right, the dependency automaton for the four operations A, B, C, and D, with respect to both precedence and alternative dependencies, respectively.

Finally, an alternative between a set of operations is modeled by an alternative dependency between each pair in the set. Thus, each of these alternative dependencies can be modeled by a two-state automaton, as described above, and included in the FSC to calculate the dependency automaton. With such a modeling approach, the derived placement transitions are alternative-valid for all types of alternatives, irrespectively if the alternative contains two or more operations.

5 Quantitative studies

This section presents quantitative studies on three different manufacturing systems where the control function is modeled by operations. The aim of the studies is to correlate the number of placement transitions that are both precedence-valid and alternative-valid with different types of systems. As pointed out in Sections 3 and 4, it is enough to include these valid transitions and not the full

set of placement transitions in the restart model in order to synthesize all valid restart states.

The results are presented in Table 1. The columns show number of operations, $|\Omega|$, number of topological orderings, $|\mathbf{top}|$, the sum of precedence-valid and alternative-valid placement transitions, $|\sigma|$, and the number of placement transitions to include in the restart model without using the method presented in this paper, $|\Omega| \times 2^{|\Omega|-1}$.

In [18] a manufacturing system for processing and assembly of turbine exhaust cases for aero-engine structures are described. Since three products can be in the manufacturing system simultaneously, the model contains operation sequences in parallel. Due to the low number of precedence dependencies between the operations, many of the placement transitions are valid despite the fact that the number of operations is low.

The system studied in [17] is a more traditional assembly cell where parts are transported, fixated, and finally assembled. Most of the operations are executed according to one straight main sequence, and this main sequence contains parts where subsequences of operations are executed in parallel. The number of valid placement transitions is of the same order as for the previous system despite the fact that the number of operations is more than four times higher. The single main operation sequence is the underlying factor.

The assembly cell presented in [3] is similar to the previous system but it contains many operations with alternative dependencies, since some processes during the manufacturing can be performed by two robots. Even though the number of topological orderings are of the same order as for the first system, the number of valid placement transitions is only one third. Thus, due to the alternative dependencies, many strings in the dependency automaton will correspond to the same placement transitions.

Table 1. Results from the studies.

	$\|\Omega\|$	$\|\mathbf{top}\|$	$\|\sigma\|$	$\|\Omega\| \times 2^{\|\Omega\|-1}$
[18]	15	$4e5$	$6e3$	$2e5$
[17]	64	$1e6$	$6e3$	$6e20$
[3]	22	$5e5$	$2e3$	$5e7$

6 Conclusions

A preprocess to an existing method for offline calculation of restart states has been presented. In both the preprocess and the existing method, it is assumed that the control function in the control system is modeled by operations. The problem with the existing method is that the number of placement transitions, the transitions that model restart of the control function in the restart states, grows exponentially with the number of included operations.

This problem is attacked in this paper by preprocessing the set of placement transitions that are to be included in the existing method, by using a compact graph model to exploit precedence and alternative dependencies between the operations. Only the placement transitions that are valid with respect to these dependencies are to be included in the succeeding calculation of restart states. By adding the preprocess, the upper limit for the number of operations that can be included in the calculation of restart states can be increased.

The state space in the compact model is correlated to the dependencies between the operations. Typically more dependencies will give a smaller state space and a faster calculation of which placement transitions to include in the succeeding calculation of restart states.

For prototype calculations, the composition algorithm in Supremica[1] has been used for construction of the compact model. The valid placement transitions have thereafter been identified by a tweaked depth first search algorithm. Typically, the time for the composition is negligible in comparison to the time required by the search algorithm.

Future research originates from two questions. First, can more placement transitions be removed from the model due to other dependencies or reexecution requirements in order to unburden the succeeding calculation of restart states? Second, maybe the use of monolithic placement transitions is not the way to go. How can the placement transitions be modeled in a more modular fashion and still preserve the clear connection between error states and restart states?

References

[1] K. Andersson, B. Lennartson, and M. Fabian. Restarting Manufacturing Systems; Restart States and Restartability. *IEEE Transactions on Automation Science and Engineering*, 7(3):486–499, 2010.

[2] K. Andersson, B. Lennartson, P. Falkman, and M. Fabian. Generation of restart states for manufacturing cell controllers. *Control Engineering Practice*, 19(9):1014–1022, 2011.

[3] K. Bengtsson. *Flexible design of operation behavior using modeling and visualization*. PhD Thesis, Chalmers University of Technology, Signals and Systems, 2012.

[4] P. Bergagård and M. Fabian. Calculating restart states for systems modeled by operations using supervisory control theory. *Submitted for possible journal publication*, 2013.

[5] M. Der Jeng. Petri nets for modeling automated manufacturing systems with error recovery. *IEEE Transactions on Robotics and Automation*, 13(5):752–760, 1997.

[6] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science, 1985.

[7] I. Klein. Efficient planning for a miniature assembly line. *Artificial Intelligence in Engineering*, 13(1):69–81, 1999.

[8] D. Knuth and J. Szwarcfiter. A structured program to generate all topological sorting arrangements. *Information Processing Letters*, 2(6):153–157, 1974.

[1] A tool for formal verification and synthesis of discrete event systems. www.supremica.org

[9] 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*, 7(4):791–802, 2010.

[10] P. Loborg. Error recovery in automation - an overview. In *AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems*, 1994.

[11] P. Loborg and A. Törne. Manufacturing Control System Principles supporting Error Recovery. In *AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems*, Stanford, 1994.

[12] P. Loborg and A. Törne. Towards error recovery in sequential control applications. In *International Symposium on Robotics and Manufacturing*, pages 377–383, Montpellier, 1996.

[13] S. Miremadi, B. Lennartson, and K. Åkesson. A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata. *IEEE Transactions on Control Systems Technology*, 20(6):1421–1435, 2012.

[14] S. Mohajerani. *On Compositional Supervisor Synthesis for Discrete Event Systems*. Licentiate Thesis, Chalmers University of Technology, Signals and Systems, 2012.

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

[16] S. Shah, E. Endsley, M. Lucas, and D. M. Tilbury. Reconfigurable logic control using modular FSMs: Design, verification, implementation, and integrated error handling. In *American Control Conference*, volume 5, pages 4153–4158. American Automatic Control Council, 2002.

[17] M. R. Shoaei, B. Lennartson, and S. Miremadi. Automatic generation of controllers for collision-free flexible manufacturing systems. In *IEEE International Conference on Automation Science and Engineering*, pages 368–373, 2010.

[18] N. Sundström, O. Wigström, P. Falkman, and B. Lennartson. Optimization of Operation Sequences using Constraint Programming. In *IFAC Symposium on Information Control Problems in Manufacturing*, volume 14, pages 1580–1585, 2012.

[19] C. Syan and Y. Mostefai. Status monitoring and error recovery in flexible manufacturing systems. *Integrated Manufacturing Systems*, 6(4):43–48, 1995.

[20] A. Yalcin. Supervisory control of automated manufacturing cells with resource failures. *Robotics and Computer-Integrated Manufacturing*, 20(2):111–119, 2004.

[21] M. Zhou and F. Dicesare. Adaptive design of Petri net controllers for error recovery in automated manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):963–973, 1989.