A Dedicated Verification Approach for Scheduling in Complex Manufacturing Machines

N.J.M. van den Nieuwelaar[12], M.M.H. Driessen[3], J.F. Groote[3]

[1]ASML Netherlands B.V., De Run 6501, 5504 DR Veldhoven, The Netherlands.

[2]Department of Mechanical Engineering, Eindhoven University of Technology,

P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

[3]Division of Computer Science, Eindhoven University of Technology,

P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

N.J.M.v.d.Nieuwelaar@tue.nl, M.M.H.Driessen@student.tue.nl, J.F.Groote@tue.nl

Abstract

Dynamic scheduling of the production in complex industrial manufacturing machines can lead to problems such as, for instance, deadlocks bringing the production to a standstill. An approach to ensure that deadlock is avoided in these highly flexible systems is to investigate all potential schedules. But verification of industrially sized systems by state-space traversal is practically impossible due to combinatoric effects causing state-space explosion. We present an effective approach to overcome this problem. The approach exploits specific characteristics of the systems and properties under consideration. Three situations in which state-space reduction techniques can be applied have been identified, formally defined and implemented in a dedicated checker. Results show that two of the techniques can reduce state-spaces with an exponential factor, and one technique has a linear effect. The application on a number of wafer scanners shows that this approach makes it possible to verify industrially sized systems.

1 Introduction

The purpose of a manufacturing machine is to make products, which requires physical manufacturing processes to be carried out. In complex manufacturing machines, different types of products are concurrently being processed. Each type of product requires different manufacturing processes. To account for these recipe dependencies, a scheduling-based control approach is proposed in [14]. The manufacturing steps to be performed are referred to as tasks, whereas the parallel operating mechatronic systems are referred to as resources. Supervisory Machine Control (SMC) is responsible for deciding when to do which tasks on which resources.

In complex manufacturing machines, many options exist to deploy the available resources to perform tasks that lead to the desired manufacturing purpose. Hence, such machines will exhibit a huge number of different machine behaviors that cannot all be tested beforehand. The choices in these machines can be categorized in three areas: which tasks to do, which resources to assign to them and which sequence of tasks to do at each resource [12]. Conversely, the machine hardware imposes physical restrictions on the freedom for choices, e.g. with respect to material logistics [13]. The freedom for choices that is applicable for some manufacturing request and for some machine with its logistic restrictions can be defined in a scheduling model or system definition.

The system definition outlines the feasible behavior of the system, which still allows for invalid behavior. Non-FIFO behavior can be an example of invalid behavior: material leaves the machine in another order than it entered the machine. Another problem is deadlock. For instance a product must enter a particular machine unit that must first be emptied. But the machine unit can only be emptied along the route that the product is blocking. This is a small instance of so-called

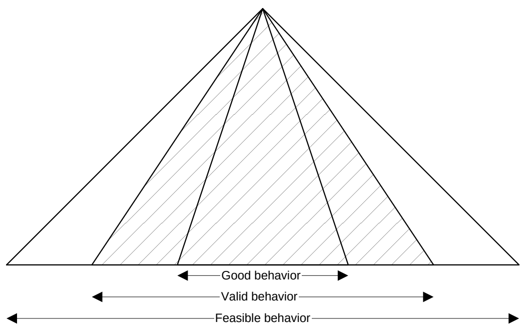


Figure 1: Search tree visualization of state-space

circular waiting situations of resources that have become a real problem in automated complex manufacturing systems.

Literature describes many approaches to avoid a manufacturing system to stop manufacturing due to deadlocks. For an extensive overview we refer to [4]. However, to the best of the authors' knowledge, the expressivity of their modelling approach is too limited for complex manufacturing machines. Some consider only a fixed routing per product [2, 16, 17, 18], or at least a predefined sequence of manufacturing processes [15]. In complex manufacturing machines, the manufacturing sequence can be flexible and even choices with respect to which manufacturing processes to carry out may be present. The scheduling-based modelling approach presented in [12, 13] captures this scheduling flexibility. The deadlock avoidance techniques used in [2, 16, 17, 18] cannot be proven to be correct in the more flexible domain of complex manufacturing machines. Also deadlock resolution as described in [19, 21, 22] is not desirable as deadlocks may not be locally resolvable due to lack of buffer places. As a consequence, we have to develop another approach to verify deadlock absence in [12, 13].

In [13], validity constraints are described to avoid invalid behavior, especially deadlocks. First, constraints on the number of material instances residing at a set of resources are defined, like in [2]. These are called *maxWIP constraints*. Second, *tied precedences* are formulated, which express that tasks must be executed after each other without being interrupted, like in [18]. These constraints are especially formulated in this way, because they can easily and very quickly be checked. The scheduler must only verify that each action that it prescribes to the machine will never lead to a situation that invalidates a constraint.

The purpose of this paper is to show how to verify that validity constraints indeed always avoid invalid machine behavior. This paper concentrates on the deadlock absence property.

SMC uses guiding heuristics to quickly find a good schedule with respect to timing performance. Such heuristics compare the time effects of scheduling alternatives. In Fig. 1 the possible schedules are depicted as a search tree. The total tree covers all schedules representing physically feasible behavior (A). The validity constraints restrict this space to result in the space of valid behavior (B), which in turn is restricted by guiding heuristics to result in the space of good behavior (C). As mentioned before, the purpose of this paper is to verify that SMC avoids invalid behavior, more specifically deadlocks. It may seem to be sufficient to explore area C for deadlocks. However, in practice it is not adequate to check area C. The guiding heuristics use predicted timing information that in the real-time environment of SMC can differ from the actual timing. This difference can trigger a rescheduling action, in which the guiding heuristics might choose another schedule due to the different timing. As a consequence, area C can drift. However, it will always stay within area B. Therefore, area B must be explored on the basis of the system definition and the validity constraints.

It is technically convenient to use a directed graph, also called a *state space* or *transition system*, to represent the behavior of complex manufacturing machines. In essence, checking for the absence of deadlocks boils down to verifying whether each state has at least one outgoing edge until manufacturing is finished. Due to several effects, the state-space can grow exponentially, which is called *state-space explosion.* This is partly due to parallelism in the system. If in some state n tasks can be executed in parallel, interleaving implies $n!$ possible execution sequences. Other exponential effects originate from the combinatoric structure of the scheduling problem. For instance m equivalent resources may be available in the machine, which is for instance the case in the Generalized Job Shop scheduling problem [20]. If a product visits such a resource n times, m^n different schedules are possible. Another combinatoric scheduling example is a set of n tasks having no precedence relation that must be executed by the same resource. These tasks can be scheduled in $n!$ different sequences, like in the Travelling Salesman scheduling problem [10].

These exponential effects often make it practically impossible to perform an exhaustive state-space traversal for industrially sized problems. One approach to limit the state-space explosion is to manually tailor the abstraction level of the model to the property to be verified, and prove that the different models that are used for the different properties are behaviorally consistent, as in [8].

In this paper, we take a radically different approach. We have developed a general tool into which we can feed system definitions and validity constraints and which outputs whether the constraints are sufficient to guarantee absence of deadlock. We use our specific knowledge about the nature of manufacturing machines to make reductions of the state-space in such a way that bad behavior is preserved and can be detected in the reduced state-space.

The first reduction that we apply is to give actions priority and ignoring the other actions in certain states. This can be justified using a confluence argument [7]. The effect of this operation is that for independent tasks only one particular order needs to be investigated in which they can occur. All other permutations of this ordering can be ignored, which is similar to partial order reduction [5].

The second reduction is based on the symmetry between equivalent resources that have the same material capacity and the same flow of material. For any machine model, it does not matter via which resource products are processed. Technically spoken, both options are strongly bisimilar [11], and it is allowed to explore only one.

The third reduction is based on the irrelevance of the scheduled sequence of certain tasks that have no logistic effect for the deadlock absence property. In this case a confluence argument can be used as well, to show that it is sufficient to explore only one schedule to investigate all possible deadlocks.

Depending on the particular circumstances, the reduction techniques can reduce the state-space exponentially, preserving the deadlocks that may be present in the behavior of the machine. Using manufacturing facilities with different layouts we show the effect of the reductions. The tool is currently in use to analyze the designs of various wafer scanners and regularly finds unexpected deadlock situations. At the end of the paper we show its applicability using two of such examples.

The structure of this paper is as follows. In Section 2, a manufacturing system is defined using a static and a dynamic system definition. The behavior of such systems is defined as a transition system. Based on this transition system the validity constraints as well as the deadlock property are defined. Section 3 describes three specific cases in which state-space reduction techniques can be applied. The reduction effect of each of the three reduction techniques is illustrated using typical example manufacturing systems in Section 4. Moreover, several instances of a wafer scanner from the semiconductor industry ([1]) are used to show the applicability of the approach in industrial practice. Finally, concluding remarks are presented in Section 5.

2 Definition of the scheduling model as a transition system

We use the manufacturing model from [12, 13]. This model consists of a static part and a dynamic part. The static part defines the machine-specific restrictions imposed by the hardware of the machine, which is resource related. It describes which capabilities the resources can provide, how many material instances can reside on them, and which logistic transports are possible.

Definition 2.1 (Static system definition) *A static system definition is a 5-tuple* $\Sigma = (R, C, A, R_m, M_f)$, *where*

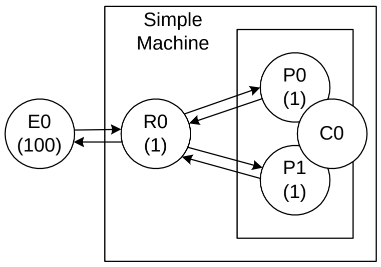


Figure 2: Layout of a simple manufacturing machine

- *R is a given set of available resources;*
- *C is a given set of capabilities;*
- *$A : R \to C$ is a function that gives the capability which a resource can provide;*
- *$R_m : R \to \mathbb{N}$ is a function that gives the maximum number of material instances that can reside on a resource, called the material capacity;*
- *$M_f : R \to \mathcal{P}(R)$ is a function that gives the resources to which material can be transported from a certain resource.*

A simple example called *Simple Machine* is depicted in Fig. 2. The machine consists of 2 equivalent main processing resources: P0 and P1. Furthermore, products can be transported from the environment (E0) to a processing resource using a robot (R0). At the processing units, a cleaning resource (C0) is available to clean products before processing. Note that although the cleaning unit is essential in the production process, products will never be put on the cleaning resource itself. The robot and the processing resources can each contain one product (denoted between brackets in Fig. 2). The arrows in Fig. 2 depict how products can be transferred from resource to resource. The static system definition of *Simple Machine* looks as follows:

- $R = \{\mathsf{E0}, \mathsf{R0}, \mathsf{P0}, \mathsf{P1}, \mathsf{C0}\}$;
- $C = \{\mathsf{E}, \mathsf{R}, \mathsf{P}, \mathsf{C}\}$;
- $A = \{(\mathsf{E0}, \mathsf{E}), (\mathsf{R0}, \mathsf{R}), (\mathsf{P0}, \mathsf{P}), (\mathsf{P1}, \mathsf{P}), (\mathsf{C0}, \mathsf{C})\}$;
- $R_m = \{(\mathsf{E0}, 100), (\mathsf{R0}, 1), (\mathsf{P0}, 1), (\mathsf{P1}, 1), (\mathsf{C0}, 0)\}$;
- $M_f = \{(\mathsf{E0}, \{\mathsf{R0}\}), (\mathsf{R0}, \{\mathsf{E0}, \mathsf{P0}, \mathsf{P1}\}), (\mathsf{P0}, \{\mathsf{R0}\}), (\mathsf{P1}, \{\mathsf{R0}\})\}$.

The dynamic system definition describes the tasks to do for a certain manufacturing request. They can recursively be structured in clusters and groups [12]. The set of all tasks, clusters and groups are the nodes. A cluster indicates that if the cluster node is chosen all children of the cluster must be chosen, whereas a group has an attribute defining the allowed numbers of children to be chosen. Such an allowed number can be less than the number of children, which makes it possible to bypass tasks. Furthermore, as defined later, precedences of nodes cannot cross group boundaries, but they can pass over cluster boundaries.

For each task the set of involved capabilities is defined, and at which capability which material instances reside at the beginning and at the end of the task. This, together with the precedence relation between nodes outlines the room for choices with respect to task order.

Definition 2.2 (Dynamic system definition) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition. Then a dynamic system definition is a 12-tuple $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$, where*

- *T is a given set of tasks;*
- *G is a given set of groups;*
- *L is a given set of clusters;*

  Now we define N, the set of nodes, as: $N = T \cup G \cup L$

- $L_n : L \to \mathcal{P}(N)$ *gives the nodes contained in a certain cluster;*
- $G_n : G \to \mathcal{P}(N)$ *gives the nodes contained in a certain group;*
- $G_a : G \to \mathcal{P}(\mathbb{N})$ *gives the allowed numbers of node alternatives to be selected from a group;*
- $I : T \to \mathcal{P}(C)$ *is a function that gives the capabilities required to perform a certain task;*
- $P : N \to \mathcal{P}(N)$ *is a function that gives the predecessors of a certain node;*
- M *is the set of material instances;*
- $C_b, C_e : T \times C \to \mathcal{P}(M)$ *are functions that give the material instances residing on a resource with a certain capability that is involved in a certain task at the beginning and end of that task, respectively;*
- $\hat{m}_s : R \to \mathcal{P}(M)$ *is a function that gives the material instances initially residing on a certain resource.*

We only consider static and dynamic system definitions that satisfy the following properties:

1. The nodes in the system have a hierarchical structure. For all nodes $n \in N$ it must hold that $n \notin anc(n)$. Here, anchestor function $anc : N \to \mathcal{P}(N)$ gives the nodes in which a certain node is contained. The set $anc(n)$ is the smallest set satisfying the following conditions:
   - if $n \in G_n(n')$ or $n \in L_n(n')$, then $n' \in anc(n)$;
   - if $n'' \in anc(n')$ and $n' \in anc(n)$ then $n'' \in anc(n)$;
2. No group has only 0 as allowed number. I.e. $G_a(g) \neq \{0\}$ for all groups $g \in G$;
3. The capabilities in $C_b(t)$ and $C_e(t)$ exist also in $I(t)$. I.e. for all tasks $t \in T$ and capabilities $c \in C$ if $C_b(t,c) \cup C_e(t,c) \neq \emptyset$ then $c \in I(t)$;
4. P contains no cycles. For each node $n \in N$ it must hold that $n \notin allsucc(n)$. The function $allsucc : N \to \mathcal{P}(N)$ gives *all* successors of a node. It is the smallest set satisfying the following condition:
   - for all nodes $n' \in N$, $n'' \in \{n\} \cup anc(n)$, if $n'' \in P(n')$ then $n' \in allsucc(n)$, and $allsucc(n') \subseteq allsucc(n)$;
5. There is no precedence relation between node alternatives in a group. For all groups $g \in G$ and nodes $n \in N$ it holds that if $n \in G_n(g)$ then $P(n) = \emptyset$;
6. Precedence relations do not cross group boundaries. For all groups $g \in G$ and nodes $n, n' \in N$ it is the case that if $g \in anc(n)$ then $n' \in P(n) \Rightarrow g \in anc(n')$ and $n \in P(n') \Rightarrow g \in anc(n')$;
7. All tasks in a group concern the same material:
$$\forall t, t' \in T, g \in G \cap anc(t) \cap anc(t').\ mat(t) = mat(t').$$
   Function mat gives the materials involved with a node:
$$mat(n) = \{m \in M \mid \exists t \in T, c \in C.\ n \in \{t\} \cup anc(t) \ \wedge\ m \in C_b(t,c) \cup C_e(t,c)\};$$
8. The subsets of material instances involved in a task remain the same from the beginning to the end of a task. I.e. for all tasks $t \in T$:
$$\bigcup_{c \in I(t)} C_b(t,c) = \bigcup_{c \in I(t)} C_e(t,c).$$
   This constraint implies that only closed systems are considered where no material enters or leaves the system;
9. Initially, the material capacity of all resources is not exceeded. For any resource $r \in R$ it holds that $\#\hat{m}_s(r) \leq R_m(r)$. Here $\#S$ gives the number of elements in the set S.

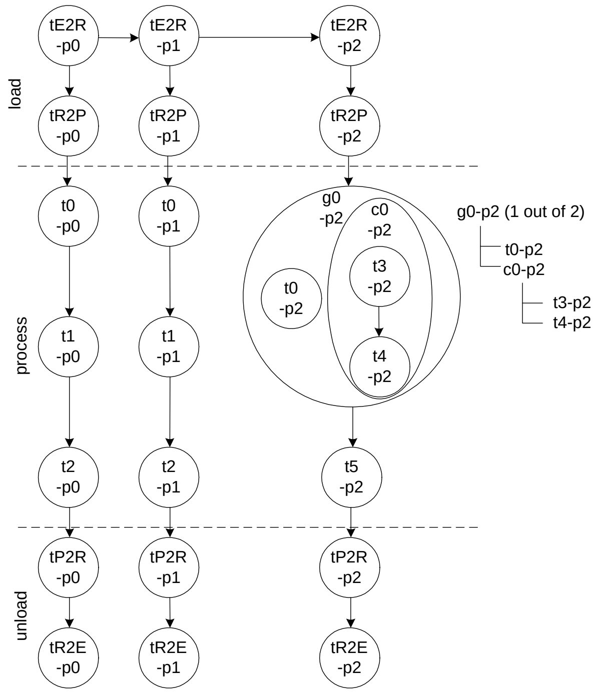


Figure 3: Three products to be produced by the simple manufacturing machine

As an example, we consider three products (p0, p1 and p2) to be manufactured by *Simple Machine*. The work to be done for these products is depicted in Fig. 3. By convention, node names end with a hyphen and the product id. Each product must be loaded, processed, and unloaded, respectively. Loading consists of an E2R task to transfer the product from the environment onto the robot, followed by an R2P task to transfer the product from the robot onto a processing unit, which is similar for unloading. The horizontal precedence arrows show that the products are loaded in the sequence p0, p1 and then p2. Products p0 and p1 are of the same type. They require a cleaning task (t0), and after that two processing tasks after another: t1 and t2, respectively. The cleaning of product p2, however, can also be done otherwise. Instead of the cleaning task, t0, cleaning can also be done by two successive other tasks, t3 and t4, that do not require the cleaning unit. Either one of the alternatives can be scheduled. In Fig. 3 this choice with respect to cleaning tasks is depicted as nested nodes in the graph, and additionally as a hierarchical node structure at the right. After cleaning, p2 requires one process task, t5. The dynamic system definition for this example looks as follows:

- T={tE2R-p0, tE2R-p1, tE2R-p2, tR2P-p0, tR2P-p1, tR2P-p2, t0-p0, t0-p1, t0-p2, t3-p2, t4-p2, t1-p0, t1-p1, t2-p0, t2-p1, t5-p2, tP2R-p0, tP2R-p1, tP2R-p2, tR2E-p0, tR2E-p1, tR2E-p2};
- G={g0-p2};
- L={c0-p2};
- L_n={(c0-p2, {t3-p2, t4-p2})};
- G_n={(g0-p2, {t0-p2, c0-p2})};
- G_a={(g0-p2, {1})};
- I={(tE2R-p0, {E, R}), (tE2R-p1, {E, R}), (tE2R-p2, {E, R}), (tR2P-p0, {R, P}), (tR2P-p1, {R, P}), (tR2P-p2, {R, P}), (t0-p0, {P, C}), (t0-p1, {P, C}), (t0-p2, {(P, C}), (t3-p2, {P}), (t4-p2, {P}), (t1-p0, {P}), (t1-p1, {P}), (t2-p0, {P}), (t2-p1, {P}), (t5-p2, {P}), (tP2R-p0, {P, R}), (tP2R-p1, {P, R}), (tP2R-p2, {P, R}), (tR2E-p0, {P, E}), (tR2E-p1, {P, E}), (tR2E-p2, {P, E})};

- P={(tR2P-p0, {tE2R-p0}), (t0-p0, {tR2P-p0}), (t1-p0, {t0-p0}), (t2-p0, {t1-p0}), (tP2R-p0, {t2-p0}), (tR2E-p0, {tP2R-p0}), (tE2R-p1, {tE2R-p0}), (tR2P-p1, {tE2R-p1}), (t0-p1, {tR2P-p1}), (t1-p1, {t0-p1}), (t2-p1, {t1-p1}), (tP2R-p1, {t2-p1}), (tR2E-p1, {tP2R-p1}), (tE2R-p2, {tE2R-p1}), (tR2P-p2, {tE2R-p2}), (g0-p2, {tR2P-p2}), (t4-p2, {t3-p2}), (t5-p2, {g0-p2}), (tP2R-p2, {t5-p2}), (tR2E-p2, {tP2R-p2})};
- M={p0, p1, p2};
- C_b={((tE2R-p0, E), {p0}), ((tE2R-p0, R), ∅), ((tE2R-p1, E), {p1}), ((tE2R-p1, R), ∅), ((tE2R-p2, E), {p2}), ((tE2R-p2, R), ∅), ((tR2P-p0, R), {p0}), ((tR2P-p0, P), ∅), ((tR2P-p1, R), {p1}), ((tR2P-p1, P), ∅), ((tR2P-p2, R), {p2}), ((tR2P-p2, P), ∅), ((t0-p0, P), {p0}), ((t0-p0, C), ∅), ((t0-p1, P), {p1}), ((t0-p1, C), ∅), ((t0-p2, P), {p2}), ((t0-p2, C), ∅), ((t3-p2, P), {p2}), ((t4-p2, P), {p2}), ((t1-p0, P), {p0}), ((t1-p1, P), {p1}), ((t2-p0, P), {p0}), ((t2-p1, P), {p1}), ((t5-p2, P), {p2}), ((tP2R-p0, P), {p0}), ((tP2R-p0, R), ∅), ((tP2R-p1, P), {p1}), ((tP2R-p1, R), ∅), ((tP2R-p2, P), {p2}), ((tP2R-p2, R), ∅), ((tR2E-p0, R), {p0}), ((tR2E-p0, E), ∅), ((tR2E-p1, R), {p1}), ((tR2E-p1, E), ∅), ((tR2E-p2, R), {p2}), ((tR2E-p2, E), ∅)};
- C_e={((tE2R-p0, E), ∅), ((tE2R-p0, R), {p0}), ((tE2R-p1, E), ∅), ((tE2R-p1, R), {p1}), ((tE2R-p2, E), ∅), ((tE2R-p2, R), {p2}), ((tR2P-p0, R), ∅), ((tR2P-p0, P), {p0}), ((tR2P-p1, R), ∅), ((tR2P-p1, P), {p1}), ((tR2P-p2, R), ∅), ((tR2P-p2, P), {p2}), ((t0-p0, P), {p0}), ((t0-p1, P), {p1}), (t0-p0, C), ∅), ((t0-p1, C), ∅), ((t0-p2, P), {p2}), ((t0-p2, C), ∅), ((t3-p2, P), {p2}), ((t4-p2, P), {p2}), ((t1-p0, P), {p0}), ((t1-p1, P), {p1}), ((t2-p0, P), {p0}), ((t2-p1, P), {p1}), ((t5-p2, P), {p2}), ((tP2R-p0, P), ∅), ((tP2R-p0, R), {p0}), ((tP2R-p1, P), ∅), ((tP2R-p1, R), {p1}), ((tP2R-p2, P), ∅), ((tP2R-p2, R), {p2}), ((tR2E-p0, R), ∅), ((tR2E-p0, E), {p0}), ((tR2E-p1, R), ∅), ((tR2E-p1, E), {p1}), ((tR2E-p2, R), ∅), ((tR2E-p2, E), {p2})};
- $\hat{m}_s$={(E0, {p0, p1, p2}), (R0, ∅), (P0, ∅), (P1, ∅), (C0, ∅)}.

Given the two system definition parts, the physically feasible behavior of the machine carrying out the schedule is defined in the form of a transition system, which is often also called a state-space or (behavioral) automaton, in Def. 2.7. The states represent which tasks have been executed and which material instances reside at which resources. The transitions indicate all allowed possibilities to go from one such state to another.

Before giving the main definitions two auxiliary concepts must be characterized. The predicate *successful*(n, tp) expresses whether node n is successfully executed given the successful tasks in tp. A task is successful if it occurs in tp. A cluster is successful if all its subnodes are successful. A group is successful iff an allowed number of nodes in the group are successful. If this number is zero, all predecessors of the group need to be successful.

Definition 2.3 *Let $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $n \in N$ be a node and $tp \subseteq T$ a set of executed tasks. We inductively define that n is successful in tp, notation successful(n, tp), iff*

- *if $n \in T$ is a task, then $n \in tp$;*
- *if $n \in L$ is a cluster, then for all $n' \in L_n(n)$ it must hold that successful(n', tp);*
- *if $n \in G$ is a group, then*

$$\#\{n' \in G_n(n) \mid \mathit{successful}(n', tp)\} \in G_a(n) \wedge$$
$$(\#\{n' \in G_n(n) \mid \mathit{successful}(n', tp)\}) = 0 \Rightarrow \forall n'' \in P(n).\ \mathit{successful}(n'', tp).$$

For a set of nodes np and a set of executed tasks tp the predicate successful(np, tp) holds if for all nodes $n \in np$ successful(n, tp) is valid.

The second auxiliary predicate that we define is *bypassed*. In order to define it, we must introduce two additional predicates. For a node $n \in N$ the expression *successor*(n) gives the successor nodes of n that can immediately be executed after n. The definition is somewhat involved, because it can be that a direct successor of n is a group in which 0 tasks can be executed. Then the successor of such a group is also a successor of n.

Definition 2.4 *Let $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. The predicate successor(n) for a node $n \in N$ is defined as the smallest predicate satisfying:*

$$\begin{array}{rcl} \mathit{successor}(n) & = & \{n' {\in} N \mid (n \cup \mathit{anc}(n)) \cap P(n') \neq \emptyset\} \cup \\ & & \{\mathit{successor}(n') \mid n' {\in} \mathit{successor}(n) \cap G \ \wedge \ 0 {\in} G_a(n')\}. \end{array}$$

Another additional definition is the notion of *initiated* nodes. These are nodes that have already been started.

Definition 2.5 *Let $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. For a set of executed tasks tp the set of nodes initiated(tp) is defined by*

$$\mathit{initiated}(tp) = \{n {\in} N \mid \exists t {\in} T.\ t {\in} tp \ \wedge \ n {\in} \mathit{anc}(t)\}.$$

The second auxiliary predicate that we define is *bypassed*(t, tp) for a task $t \in T$ and a set of tasks $tp \subseteq T$. Its definition is quite involved. The predicate *bypassed*(t, tp) holds for task t and set of tasks tp, if t is not successful (i.e. $t \notin tp$) and

- either a subsequent task is already successful,
- or the number of initiated nodes in a group containing that task is equal to the maximum number of nodes that can be successful in that group.

Definition 2.6 *Let $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $t \in T$ be a task and $tp \subseteq T$ a set of executed tasks. We define that t is bypassed, notation bypassed(t, tp) iff*

$$\begin{array}{l} t \notin tp \wedge \exists g {\in} G \cap \mathit{anc}(t). \\ \qquad \mathit{successor}(g) \cap tp \neq \emptyset \ \vee \\ \qquad\ ((\mathit{anc}(t) \cup \{t\}) \cap G_n(g)) \setminus \mathit{initiated}(tp) \neq \emptyset \ \wedge \\ \qquad\qquad \#(G_n(g) \cap \mathit{initiated}(tp)) = \max(G_a(g)). \end{array}$$

Here $\max(S)$ *for a finite set of natural numbers S is the largest number in S. The set bypassed(tp) is defined as $\{t {\in} T \mid \mathit{bypassed}(t, tp)\}$.*

This provides sufficient basic material to define the state-space of a system.

Definition 2.7 (The state-space of the system) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Then the state-space is defined as the transition system $\Omega = (St, \hat{s}, Ac, \tau)$, where*

- $St = (\mathcal{P}(T) \times (R \to \mathcal{P}(M)))$ *are the states. The first component indicates the tasks that have been executed and the second component indicates which material instances can be found at which resource.*
- $\hat{s} = (\emptyset, \hat{m}_s)$ *is the initial state;*
- $Ac = (T \times \mathcal{P}(R))$ *are the labels of states indicating which task is executed at which resources;*
- $\tau = \{((tp, ms), (t, r), (tp', ms')) \in St \times Ac \times St$

$$
\begin{array}{lll}
| & \mathit{successful}(P(t), tp) & (1)\\
\wedge & t \in T\backslash(tp \cup \mathit{bypassed}(tp)) & (2)\\
\wedge & \forall cap{\in}I(t).\ \exists res{\in}r.\ A(res) = cap & (3)\\
\wedge & \forall res{\in}r.\ C_b(t, A(res)) \subseteq ms(res) & (4)\\
\wedge & \forall res{\in}r, m{\in}M. & (5)\\
 & \quad m{\in}ms(res) \Rightarrow\ m{\in}ms'(res) \vee (\exists res'{\in}M_f(res).\ m{\in}ms'(res')) & \\
\wedge & \forall res{\in}R.\ \#ms'(res) \leq R_m(res) & (6)\\
\wedge & tp' = tp \cup \{t\} & (7)\\
\wedge & \forall res{\in}r.\ ms'(res) = (ms(res)\backslash C_b(t, A(res))) \cup C_e(t, A(res)) & (8)\\
\wedge & \forall res{\notin}r.\ ms'(res) = ms(res)\} & (9)
\end{array}
$$

contains the transitions from state to state.

To make sure that the transitions conform the intuition of the dynamic system definition, several cases must be distinguished. Cases one through three are properties that must hold for the task that is performed in the transition and cases four through seven are properties that must hold for the resources assigned to the task. Seven through nine express the update of the system state. Below each line of the definition of a state-space is explained separately.

1. All predecessors of the task that is performed in the transition were successful before the transition.
2. The task to be performed did not happen and has not been bypassed.
3. For every capability involved in the task, there is a resource assigned to the task that can provide that capability.
4. The material to be used in the task is present at the resources that are assigned to the task. This check imposes *logistic flow integrity* as defined in [13].
5. Material that is at an involved resource before a transition is at that same resource after the transition, or it has been transported to a resource to which it could be transported. This check imposes *logistic flow feasibility* as defined in [13].
6. After a transition, the material capacity of all resources is not exceeded. This check imposes *material capacity feasibility* as defined in [13].
7. The set tp' contains all tasks that have been performed before the transition and the task that has been performed during the transition.
8. After the transition, the material configuration of the resources involved in the task is the same as the material configuration at the beginning of the transition except for the changes made by the task.
9. After the transition, the material configuration of the resources not involved in the task is the same as the material configuration at the beginning of the transition.

As already argued in the introduction, it is possible that deadlocks occur in the state-space. In the simple example, deadlock occurs when three products are loaded without one being unloaded first, as is depicted in Fig. 4. The product at resource `R0` wants to move to either `P0` or `P1`, whereas the products at resources `P0` or `P1` must move to `R0`.

In order to avoid such undesirable deadlocks, validity constraints are defined. A validity constraint is a predicate on states that expresses which states are and which are not accessible. The intention is that the validity constraints are defined such that deadlocks are avoided. Furthermore, these constraints are defined such that they can be quickly checked while dynamically scheduling a next task. We first define a constraint in an abstract sense.

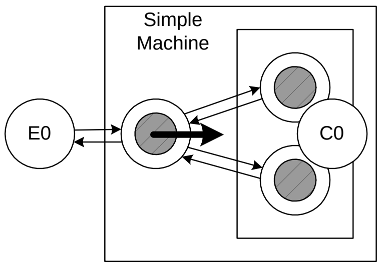


Figure 4: Deadlock in the simple manufacturing machine

Definition 2.8 *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $\Omega = (St, \hat{s}, Ac, \tau)$ be a state-space. We call a function $VC : \tau \to \mathbb{B}$ a validity constraint. The constrained state-space $\Omega_{VC} = (St, \hat{s}, Ac, \tau_{VC})$ has a constrained transition relation which is defined by*

$$\tau_{VC} = \{(s, (t, r), s') \in \tau \mid VC((s, (t, r), s'))\}.$$

Note that the trivial validity constraint $VC(s) = true$ for any $s \in \tau$ keeps the original state-space unchanged.

We explicitly define the notion of a deadlock state in a constrained state space as those states that do not have outgoing edges, but are also not finished. A finished state is a state where all tasks are either successful or bypassed.

Definition 2.9 (Finished and deadlocked states) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $\Omega = (St, \hat{s}, Ac, \tau)$ be a state-space and $VC : \tau \to \mathbb{B}$ be a validity constraint. The set of finished states $F(\Omega_{VC})$ is defined as*

$$F(\Omega_{VC}) = \{(tp, ms) \in St \mid (\forall t \in T.\ \exists n \in \{t\} \cup anc(t).\ successful(n)\}.$$

The set of deadlock states $D(\Omega_{VC})$ is defined by:

$$\begin{aligned} D(\Omega_{VC}) = \{(tp, ms) \in St \mid\, & (tp, ms) \notin F(\Omega_{VC}) \wedge \\ & \forall t \in T, r \in \mathcal{P}(R), (tp', ms') \in St.\ ((tp, ms), (t, r), (tp', ms')) \notin \tau_{VC}\}. \end{aligned}$$

Note that a state-space of a production system never has a loop. This follows from the fact that in each transition the set *tp* is always extended with one task. A consequence of this is that a constrained state-space without deadlocks always has finished states that will be reached. In other words, a constrained production machine works fine as long as deadlock freedom has been shown. In essence checking deadlock freedom is very easy by inspecting each state. The only problem is that the number of states can grow dramatically. In the next section techniques are provided to reduce the size of the state-space maintaining the deadlocks in the system.

We define two concrete classes of validity constraints. The first one restricts the maximum number of material instances that are allowed to occupy a group of resources. These are called *maximum work in progress* constraints or *maxWIP* constraints. For the simple machine the maxWIP constraint that expresses that at most two material instances may occupy R0, P0 and P1 is sufficiently strong to prevent deadlocks. These constraints are formulated using the predicate $maxWIP_\Gamma$ where Γ is a set of pairs of the form (r, n). Here $r \subseteq R$ is a set of resources and $n \in \mathbb{N}$ is a natural number. It is formalized as follows:

Definition 2.10 (MaxWIP constraints) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition, $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ a dynamic system definition and $\Omega = (St, \hat{s}, Ac, \tau)$*

a state-space. Let Γ be a set of pairs with a set of resources and a natural number. Then $maxWIP_\Gamma$ is the validity constraint defined as:

$$maxWIP_\Gamma(((tp, ms), (t, r), (tp', ms'))) = \bigwedge_{(r,n)\in\Gamma} (\sum_{res\in r} \#ms'(res)) \leq n$$

for every transition $((tp, ms), (t, r), (tp', ms'))) \in \tau$.

Another way to avoid deadlocks is to use precedence constraints. The idea is that some tasks, or groups of tasks, can only be executed if it is guaranteed that another task can succeed it. The first task is called a tied predecessor of the second.

In the example of the simple production machine deadlock can be avoided by tying the movements of products onto the robot to the task of moving the product from the robot. In this way the robot will always become empty and available to move further products.

We use a function $P_t : N \to N \cup \{\perp\}$ that provides the tied predecessor of a node, unless there is no such a tied predecessor, in which case it yields $\perp$. We assume that P_t satisfies that if $P_t(n) = P_t(n')$ and $P_t(n) \neq \perp$, then $n = n'$.

Definition 2.11 (Tied precedence constraints) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $\Omega = (St, \hat{s}, Ac, \tau)$ be a state-space. Let $P_t : N \to N$ be a tied predecessor function satisfying that $P_t(n) \in \bigcup_{m\in M} P_m(n) \cup \{\perp\}$ for all $n \in N$. Here, P_m is the part of P that concerns material m:*

$$P_m(n) = \{n' \in P(n) \mid m \in mat(n) \cap mat(n')\}.$$

Then the tied precedence constraint $tiedprecedence_{P_t}$ is defined by:

$$\begin{aligned} &tiedprecedence_{P_t}(((tp, ms), (t, r), (tp', ms'))) = \\ &\qquad \forall t' \in tp.\ (t \in ftsucc(t', P, P_t) \vee ftsucc(t', P, P_t) = \emptyset) \wedge carrythrough((tp', ms')). \end{aligned}$$

We define $ftsucc(n, P, P_t)$ as the first tasks in a tied successor node of n:

$$ftsucc(n, P, P_t) = \{n'' \in firsttasks(n') \mid n' \in succ(n, P) \cap succ(n, P_t)\}.$$

Here, $succ(n, V)$ gives the successors of a node n, given a precedence relation V:

$$\begin{aligned} succ(n, V) = &\{n' \in N \mid n \in V(n')\} \cup \\ &\{n'' \in N \mid \neg\exists n' \in N.\ n \in V(n') \wedge \\ &\qquad \exists n''' \in N.\ n \in G_n(n''') \cup L_n(n''') \wedge n'' \in succ(n''', V)\}. \end{aligned}$$

If no nodes succeed n directly, a node one level higher in the node hierarchy is looked at. Note that due to the hierarchical node structure, succ is well defined.

The function firsttasks gives the first tasks in a node:

$$firsttasks(n) = \{t \in T \mid n \in \{t\} \cup anc(t) \wedge \neg\exists n' \in N.\ t \in allsucc(n') \wedge n \in anc(n')\}.$$

The predicate carrythrough indicates that all tied successors of a task can be finished.

$$\begin{aligned} &carrythrough((tp, ms)) = \\ &\qquad \forall t \in tp, t' \in T.\ t' \in ftsucc(t, P, P_t) \wedge \neg successful(t') \Rightarrow \\ &\qquad\quad \exists r \in \mathcal{P}(R), tp' \in \mathcal{P}(T), ms' \in R \to \mathcal{P}(M). \\ &\qquad\qquad ((tp, ms), (r, t'), (tp', ms')) \in \tau \wedge carrythrough((tp', ms')). \end{aligned}$$

Note that as the state-space has no cycles, carrythrough is well defined.

3 Checking deadlocks by reducing the state-space

In order to know that maxWIP and tied precedence constraints guarantee that it is impossible to reach deadlock states, it is sufficient to generate all possible states of the machine that can be reached and to check that each such state is not a deadlock state (Def. 2.9).

This can very simply be achieved, except that the number of states in the machine is generally (larger than) astronomical. In this section, we present three ways to reduce the size of the state-space, while maintaining potential deadlocks. For detailed proofs we refer to [3].

3.1 Interleaving of independent parallel tasks

The first reduction technique has to do with interleaving of independent parallel tasks. In some situations, no matter what interleaving is chosen during traversal, the same state will be encountered in the end. A general state-space reduction technique that can be applied in this case is the priorisation of actions [7], also called τ-priorisation. In literature, τ is used to denote internal transitions, which applies to all our transitions, because we are interested in deadlocks, and not in the particular nature of individual transitions. It is allowed to apply priorisation of actions, if the system has no infinite behavior and is confluent. Our transition system has no infinite behavior, as with each step, an additional task is executed. Confluence is defined below. It says that in every state, where a transition from a confluent set of transitions can be chosen, and another transition is possible, a common state can be reached. The definition of confluence is given in the form of a set of conditions on a confluent set of transitions (cf. [6]).

Definition 3.1 (Confluence) *Let $(St, \hat{s}, Ac, \tau)$ be a state-space. Let $\mathbf{C} \subseteq \tau$ be a set of transitions. The set $\mathbf{C}$ is called τ-confluent if for all $(s, a, s') \in \mathbf{C}$ and $(s, a', s'') \in \tau$ it holds that:*

$$(\exists s''' \in St.\ (s', a', s''') \in \tau \wedge (s'', a, s''') \in \mathbf{C}) \ \vee \tag{10}$$

$$\exists a'' \in Ac.\ (s'', a'', s') \in \mathbf{C} \ \vee \tag{11}$$

$$\exists a''' \in Ac.\ (s', a''', s'') \in \tau \ \vee \tag{12}$$

$$s' = s'' \tag{13}$$

The prioritized state-space is defined as follows. It says that Ω' can be constructed out of Ω by removing outgoing transitions from a state, as long as at least one transition in $\mathbf{C}$ remains. This generally reduces the size of a transition system substantially, especially, because many states become unreachable. We have the important theorem that says that if we apply τ-priorisation with a τ-confluent set, then the state-space of the system has *exactly* the same deadlocks.

Definition 3.2 (τ-priorisation) *Let $\Omega = (St, \hat{s}, Ac, \tau)$ and $\Omega' = (St, \hat{s}, Ac, \tau')$ be state-spaces. Let $\mathbf{C} \subseteq \tau$ be a set of transitions. We say that Ω' is a τ-prioritized reduction of Ω iff*

- $\tau' \subseteq \tau$;
- $\forall s, s' \in St, a \in Ac.\ (s, a, s') \in \tau \ \Rightarrow\ (s, a, s') \in \tau' \vee \exists s'' \in St, a' \in Ac.\ (s, a', s'') \in \tau \cap \mathbf{C}$.

A set of τ-confluent transitions $\mathbf{C}$ in the state-space of a system is defined as follows. A more detailed explanation is given after the definition.

Definition 3.3 (Confluent transition set C) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $(St, \hat{s}, Ac, \tau)$ be the state-space of the system. We define $\mathbf{C} \subseteq \tau$ to be the set consisting of the transitions $((tp, ms), (t, r), (tp', ms'))$ for which for all transitions $((tp, ms), (t', r'), (tp'', ms''))$ with $t \neq t'$ the following conditions hold.*

1. $\forall t'' \in T.\ mat(t'') \cap mat(t) \neq \emptyset \Rightarrow (t'' \cup anc(t'')) \cap allsucc(t) \neq \emptyset$;
2. $\forall res \in r.\ res \in unsafe \Rightarrow C_e(t, A(res)) \backslash C_b(t, A(res)) = \emptyset$;

3. $\forall g \in G, n \in (g \cup anc(g)).\ (anc(t) \cup \{t\}) \cap succ(n, P) = \emptyset$*;*

4. $G \cap anc(t) = \emptyset$*.*

The auxiliary functions used in this definition are defined directly below Def. 2.2 and in Def. 2.11, except for unsafe, which is given below. The set of unsafe resources contains those resources for which it is possible to put material onto it from more than one other resource.

$$unsafe = \{res \in R \mid \sum_{res' \in R, res \in M_f(res')} R_m(res') > 1\}.$$

Condition one says that parallel tasks involving the same material are not necessarily confluent. Furthermore, the fact that in the end the same state will be encountered implies that no deadlock may be involved in the confluent set. If none of the transitions in the confluent set loads to an unsafe resource, i.e. a resource that can receive material from multiple other locations, confluence is not harmed. This is condition 2. Finally, the logistic effect of a group depends on which tasks are chosen in that group. Therefore, groups form an unreliable basis for confluence and are excluded as predecessor or relative of tasks in the confluent set. This yields conditions 3 and 4, respectively. Note that these conditions on set **C** are at the safe side. The set can be extended, but τ-confluence of an extended set has not been proven in [3].

3.2 Resource symmetry

The second reduction technique concerns equivalent resources in the system. In such a case multiple equivalent schedules can be generated, in which the only difference is that equivalent resources are swapped. The state-space can be reduced in such cases using strong bisimulation [11], provided that it does not relate finished and deadlocked states. In such a case strong bisimulation reduction maintains deadlocks in our production machines, and so, only the reduced system needs to be investigated.

Definition 3.4 (Strong bisimilarity) *Let $(St, \hat{s}, Ac, \tau)$ be a state-space. For two states $s, t \in St$ to be strongly bisimilar, there must be a strong bisimulation relation R relating s and t, i.e. sRt. A relation R is a strong bisimulation relation iff it is symmetric, and for all states s, t such that sRt it holds that:*

$$(s, a, s') \in \tau \Rightarrow \exists t' \in St.\ (t, a, t') \in \tau \wedge s'Rt'.$$

Two resources are symmetric in our transition system if they are of the same capability, have the same material capacity, and material can flow from and to the same resources. Note again that this definition can be extended, as e.g. also groups of resources can be symmetric, but this is not taken into account here.

Definition 3.5 (Symmetric resources) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition. We say that two resources $r \in R$ and $r' \in R$ are symmetric, notation $symm(r, r')$, iff*

$$\begin{gathered} A(r) = A(r') \wedge \\ R_m(r) = R_m(r') \wedge \\ M_f(r) = M_f(r') \wedge \\ \forall res \in R.\ r \in M_f(res) \Rightarrow r' \in M_f(res) \end{gathered}$$

Consider the state-space $(St, \hat{s}, Ac, \tau)$ of the system. If there are two states $(tp, ms) \in St$ and $(tp, ms') \in St$ such that $\forall res \in R.\ ms(res) = ms'(res) \vee \exists res' \in R.\ symm(res, res') \wedge ms(res) = ms'(res')$ then it holds that (tp, ms) and (tp, ms') are strongly bisimilar. Furthermore, it is easy to see that with this definition no deadlocked and finished states are related. So, this definition can then be employed in the following way. It is only necessary to investigate one state from the whole set of bisimilar states. So, in order to explore the full state-space, it is only necessary to investigate only the first state encountered from the whole bisimilar set of states.

3.3 Non-logistic tasks

The third state-space reduction deals with mutually exclusive non-logistic tasks. A non-logistic task has no logistic effect as the materials stay at the same resources. Tasks having no logistic effect can be executed in any arbitrary sequence without influencing deadlock behavior. Using a similar confluence argument as in Section 3.1 the state-space can be reduced. We use the following confluent transition set.

Definition 3.6 (Confluent transition set D) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $(St, \hat{s}, Ac, \tau)$ be the state-space of the system. We define $\mathbf{D} \subseteq \tau$ to be the set consisting of the transitions $((tp, ms), (t, r), (tp', ms'))$ for which the following condition hold.*

$$anc(t) \cap NL \neq \emptyset \wedge \forall g \in G \cap anc(t).\ g \in NL.$$

Here, the definition of the non-logistic groups and clusters, notation

$$NL = \{n \in (G \cup L) \mid \forall t \in T, res \in R.\ n \in anc(t) \Rightarrow C_b(t, A(res)) = C_e(t, A(res))\}.$$

This definition only considers groups or clusters containing only non-logistic tasks.

Using the fact that **D** is a confluent set of transitions, it is only necessary to investigate only one edge labelled with a non-logistic task in each state having such an edge. All other edges in this state can be ignored, while exactly finding all deadlocks in the system.

4 Results

This section presents the results that can be obtained by application of the reduction techniques presented in the previous section. A dedicated verification tool has been implemented in the C programming language. It traverses the state-space as defined in Def. 2.7 and takes into account validity constraints as defined in Def. 2.10 and Def. 2.11. Furthermore, the state-space reduction techniques as described in the previous section are implemented.

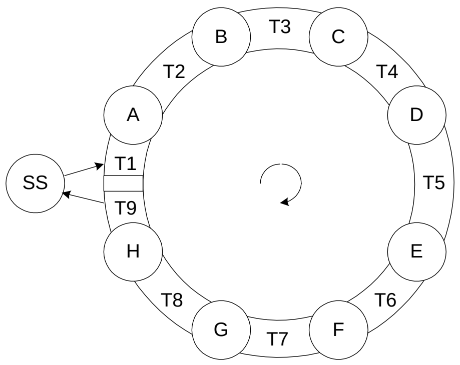
The explored system states $St = (\mathcal{P}(T) \times (R \to \mathcal{P}(M)))$ are stored such that memory usage is minimized. The states are put in a labelled structure. A bit array is used to indicate whether or not a task is passed, using which $\mathcal{P}(T)$ is implemented. Furthermore, an array of material locations is used to store the material configuration of the system, which implements $R \to \mathcal{P}(M)$. To be able to analyse an encountered deadlock state, it must be possible to generate a transition trace that led to the deadlock state. Therefore, the label of the state s that first led to a state s' is stored with state s', whereas the other transitions leading to s' are not stored at all.

This section first illustrates the reduction power of each of the three state-space reduction techniques using three typical example manufacturing systems. After that, applicability of the reduction techniques in industrial practice is illustrated using a wafer scanner [1].

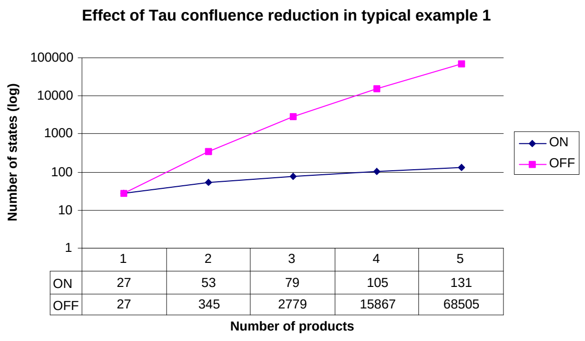
4.1 Reduction power

Interleaving of independent parallel tasks

To illustrate the power of the reduction for interleaving of independent parallel tasks, the manufacturing cell depicted in Fig. 5(a) serves as an example. The cell consists of 8 processing stations, A through H. Products enter the cell in a predefined sequence from unit SS, to which products also leave the cell. The SS unit and the processing stations are connected via a conveyor belt transportation system consisting of independent segments (T1 through T9). Each processing station and each conveyor belt segment can hold one product. Each product has to undergo 26 tasks. In case only one product is to be manufactured, 27 states are to be traversed, no matter which reduction technique is applied. In case multiple products are to be manufactured, many different interleaved traces are possible. However, interleaving of independent parallel tasks does



(a) Layout



	1	2	3	4	5
ON	27	53	79	105	131
OFF	27	345	2779	15867	68505

(b) Reduction power of interleaving reduction

Figure 5: Typical example 1: manufacturing cell

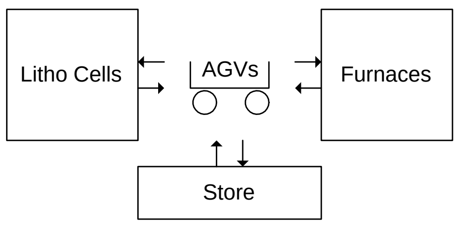
not matter for deadlock. This is where the reduction for interleaving is very powerful, as is shown in Fig. 5(b). In the graph the number of states are shown with the τ-priorisation switched on and off. In this typical case, a state-space with exponential size in the number of products is reduced to a linear state-space.

Resource symmetry

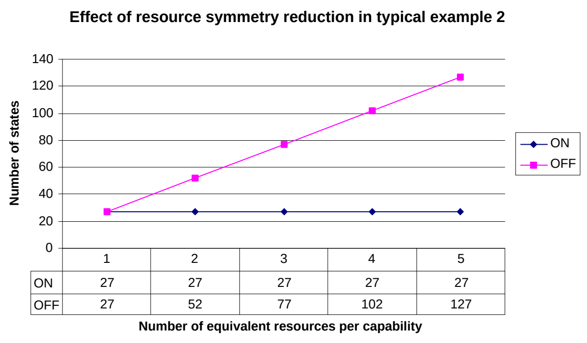
The power of the resource symmetry reduction is illustrated using an example lithographic area in a semiconductor factory, as depicted in Fig. 6(a). The area is equipped with two types of processing equipment: litho cells and furnaces. Each batch of wafers has to undergo 4 processing cycles consisting of an exposure step in a litho cell and a bake step in a furnace. The wafer batches are transported using automatic guided vehicles (AGVs). Multiple instances of the processing and transportation equipment can be available. Including the transport from and to the store, each batch of wafers has to undergo 26 tasks, like in the previous example. In case only one instance of each type of equipment is available, 27 states are to be traversed for such a batch. However, in case multiple instances (resources) of each type of equipment (capability) is available, the number of traces (or schedules) that can be followed grows rapidly. As we do not store which resources are used in the passed tasks, the effect on the growth of the state-space is only linear in the number of equipment instances. However, for the deadlock property it does not matter which instance is chosen. Reduction for resource symmetry makes the state-space size independent of the number of equipment instances, as is shown in Fig. 6(b).

Non-logistic tasks

To illustrate the power of the reduction for non-logistic tasks, the exposure of integrated circuits (ICs) onto a wafer in a wafer scanner serves as an example. During exposure, the wafer is carried by a wafer stage. Each wafer can contain over a hundred ICs. In Fig. 7(a) the center coordinates of the ICs on a typical wafer are depicted by '+' signs. SMC is free to choose in which order and in which direction (up or down) to scan the ICs such that the fastest route results. The number of routes and states is exponential in the number of ICs. However, which route is taken is not of importance for the verification of the absence of deadlock, as the wafer stays at the same resource, which is called the wafer stage. This is where the reduction for non-logistic tasks is very powerful, as is shown in Fig. 7(b). An exponential size of the state-space in the number of ICs is reduced to a linear one.



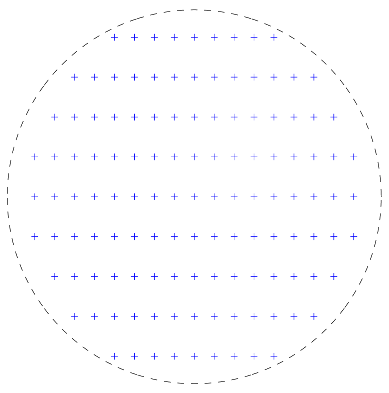
(a) Layout



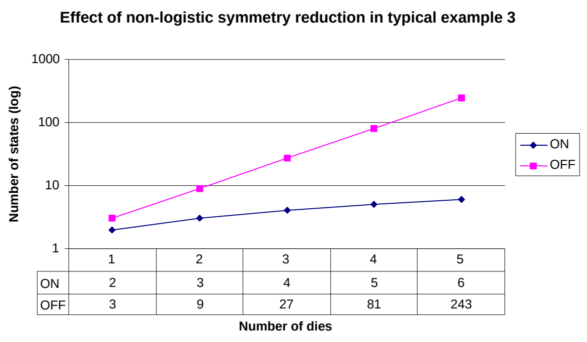
	1	2	3	4	5
ON	27	27	27	27	27
OFF	27	52	77	102	127

(b) Reduction power of resource symmetry reduction

Figure 6: Typical example 2: part of a semiconductor factory



(a) Layout



	1	2	3	4	5
ON	2	3	4	5	6
OFF	3	9	27	81	243

(b) Reduction power of non-logistic reduction

Figure 7: Typical example 3: integrated circuits on a wafer

Practical use

The applicability of the reduction techniques is illustrated using two example wafer scanners from industry. The first example concerns an existing wafer scanner, the ASML TWINSCAN. The second example concerns a wafer scanner under development at ASML that uses Extreme Ultra Violet light for exposure, which is therefore called the EUV machine.

TWINSCAN

A layout of the TWINSCAN is depicted in Fig. 8. Typical for the TWINSCAN system is that it contains two wafer stages, shown at the right of Fig. 8: WS1 and WS2. Wafers can enter and leave the system via two different resources. By default, wafers enter the machine via the track (TR), and undergo the tasks depicted at the top of Fig. 9. In this figure, tied precedence edges are labelled with a 't'. Note that the routing details as described in the third example of the previous section are left out here. It is also possible to bring wafers into the machine via the carrier handler unit (CH). In this case, the wafers can follow two paths, as depicted in the middle and the bottom of Fig. 9. As an example, we consider the processing of 3 lots, each consisting of 5 wafers (mat0 through mat14). Which path which wafer must follow is depicted at the right of Fig. 9. When

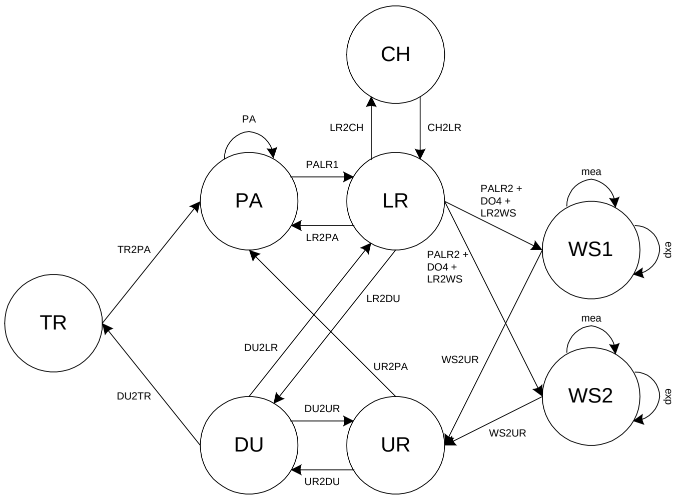


Figure 8: Layout wafer scanner with carrier handler: first industrial case

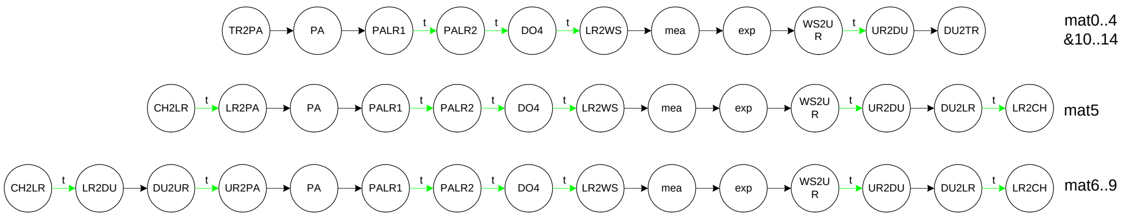


Figure 9: Different routes for first industrial case

we verify this case, we find several deadlock states. For one of them, the material configuration in the deadlock state is depicted in Fig. 10, showing a circular wait condition. An example trace that leads to this deadlock state is shown in Fig. 11. This figure contains Gantt chart bars as well as material location lines, colored per material instance. Such deadlock states can be avoided with the introduction of an additional *maxWIP* validity constraint, stating that the pre-alignment unit (PA), the wafer stages (WS1 and WS2), and the discharge unit (DU) together contain no more than three wafers. For the deadlock state in Fig. 10, mat8 would not have entered the system, and the deadlock would be avoided. Verification confirms that indeed only valid schedules remain, then. An example schedule is shown in Fig. 12.

EUV wafer scanner

A layout of the EUV wafer scanner is shown in Fig. 13. As EUV light is absorbed by air, exposure must take place in vacuum. Therefore, the machine is equipped with two load locks (L1 and L2) to bring wafers entering the machine from atmospheric pressure to vacuum, and vice versa when leaving the machine. Furthermore, instead of two single-armed robots in TWINSCAN, only one combined robot with two arms (R1a and R1b) is available for wafer transportation. Again, two wafer stages are available (WS1 and WS2). Only loading from the track (TR) is considered in this example. Three *maxWIP* validity constraints are used, as depicted by the dotted squares and the numbers in Fig. 13. The absence of tied precedences and presence of multiple resources of the machine capabilities results in many scheduling possibilities. In case of 5 wafers, the state-space consists of 67496 states. Application of the reduction techniques reduces the state-space size with one order of magnitude to 6746 states. Fig. 14 shows the influence of the number of wafers in the case on the state-space size, which still is exponential. Nevertheless, the reduction effect and

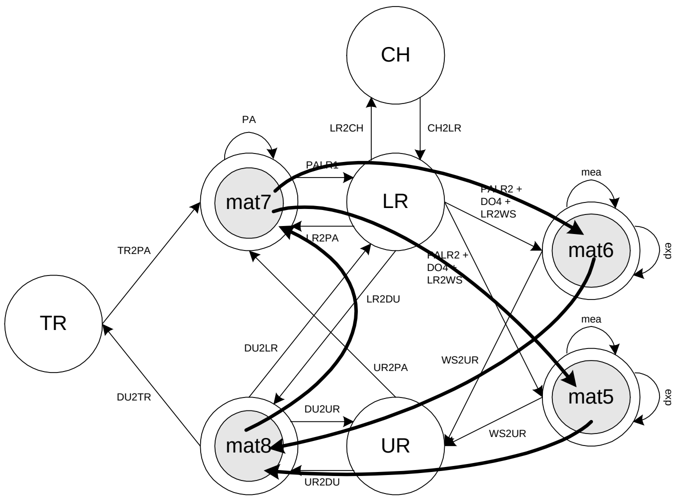


Figure 10: Deadlock state in first industrial case without maxWIP constraint

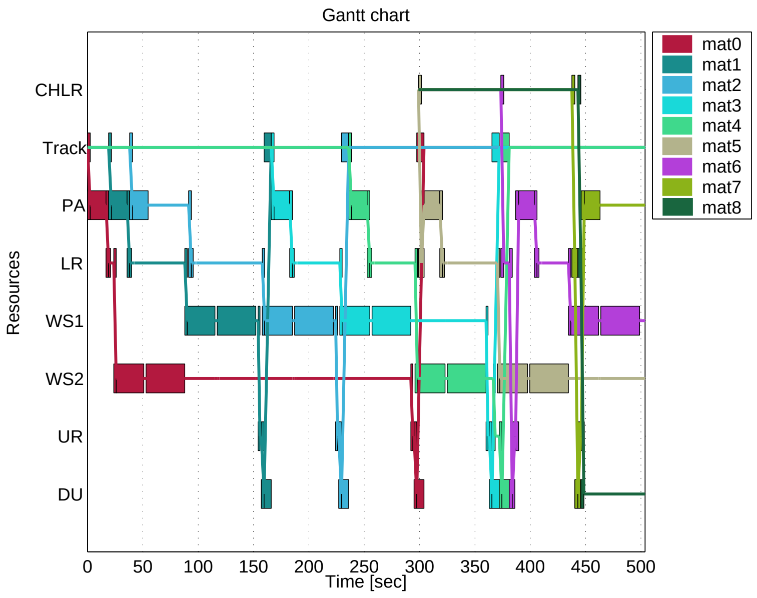


Figure 11: Deadlock trace for first industrial case without maxWIP constraint

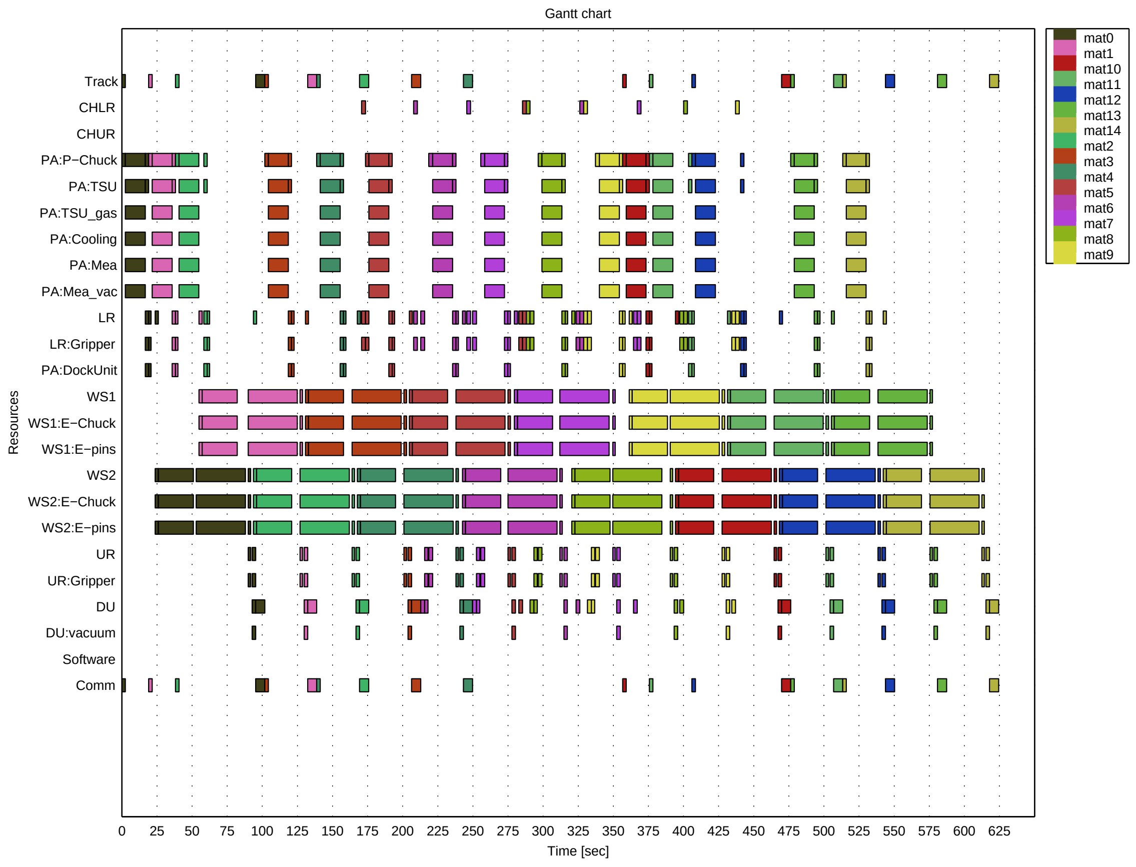


Figure 12: Schedule for first industrial case with additional maxWIP constraint

memory usage efficiency of $\mathcal{O}(10^7)$ states per gigabyte are sufficient to verify industrially sized problems.

5 Conclusions

A dedicated verification approach has successfully been developed to verify industrially sized cases of complex manufacturing machines for absence of deadlock. The cases are described by a definition of the manufacturing system (static system definition) and the work to be done (dynamic system definition), based on which a definition of the feasible state-space is defined. Furthermore, a set of validity constraints that should avoid deadlock is specified. Several combinatoric effects cause the state-space to grow exponentially, which makes it practically impossible to perform an exhaustive state space traversal for industrially sized problems.

As opposed to general paradigms to define system behavior such as automata or Petri nets, the system definition consists of lots of elements. The fact that elements have a specific and intuitive meaning makes it possible to formalize intuitive state-space reduction possibilities in terms of the system definition elements.

Three situations in which state-space reduction techniques can be applied have been identified and formally defined. They can be justified by applying meta proofs to the specific characteristics of manufacturing systems and the deadlock absence property. A verification tool has been developed that can traverse the state-space taking into account the validity constraints to detect deadlock states. The reduction techniques have been implemented, and memory usage is optimized.

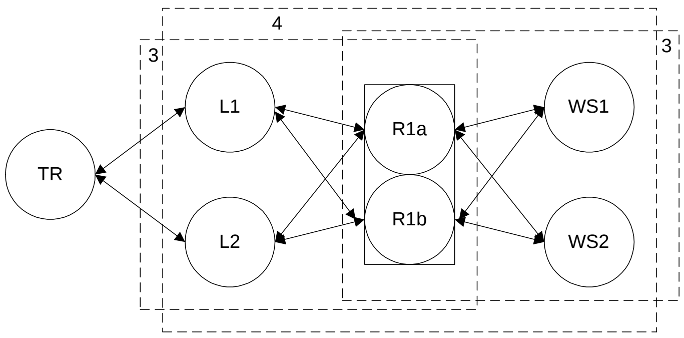


Figure 13: Layout of EUV machine: second industrial case

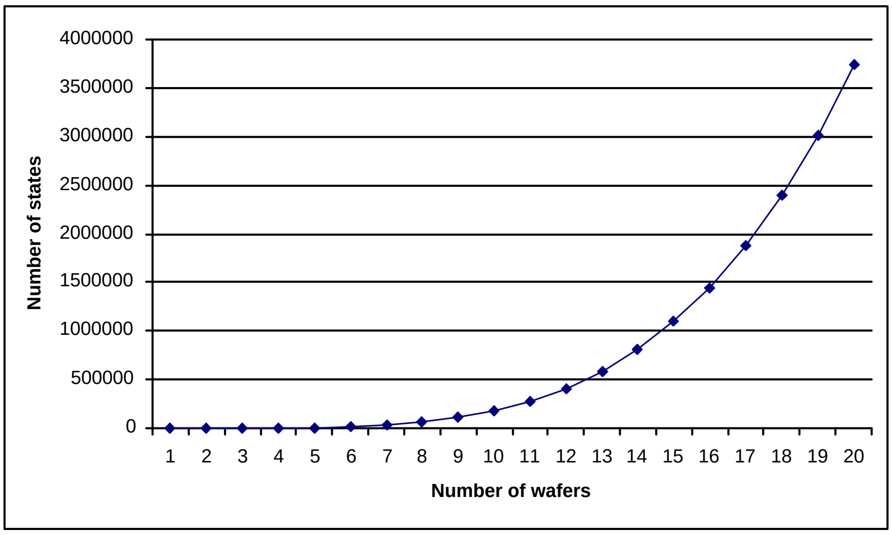


Figure 14: Number of states versus number of wafers in second industrial case

Results show that two of the techniques can reduce exponential growth to linear growth, and one technique can make the state-space size independent of a linear effect. Furthermore, results of real wafer scanner problems show that the approach is suited for verification of industrially sized problems.

The three reductions that we provide are rather straightforward. By making them more sophisticated, it should be possible to achieve further state-space reductions. Many other reductions are conceivable. An example is material symmetry. In case of identical product recipes, the state-space can consist of many recurrent patterns that only differ in terms of material instances. These recurrent patterns can be mapped onto each other, as is e.g. done in [9].

The present work shows an effective approach to overcome state-space explosion problems by using specific knowledge of the nature of the type of systems and the properties to be verified to reduce state-spaces. This approach has been used to effectively find and remove deadlock situations in a number of machines at ASML.

References

[1] ASML, 2004. Available through URL `http://www.asml.com/`.

[2] Z.A. Banaszak and B.H. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Transactions on Robotics and Automation*, 6(6):724–734, December 1990.

[3] M.M.H. Driessen. Verification of task resource scheduling. Technical report, Department of Computing Science, Eindhoven University of Technology, The Netherlands, June 2004. ASML Confidential, to be made available via home page.

[4] M.P. Fanti and M. Zhou. Deadlock control methods in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics - part A: Systems and Humans*, 34(1):5–22, 2004.

[5] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In K. G. Larsen and A. Skou, editors, *Computer Aided Verification (CAV '91)*, number 575 in LNCS, pages 332–342. Springer, 1991.

[6] J.F. Groote and J.C. van de Pol. State space reduction using partial tau-confluence. In Nielsen, Mogens, and Rovan, editors, Proceedings of Mathematical Foundations of Computer Science 2000, LNCS 1893, Springer-Verlag, pages 383–393, 2000.

[7] J.F. Groote and M.P.A. Sellink. *Confluence for Process Verification*, volume 170, pages 47–81. 1996.

[8] M. Hendriks, N.J.M. van den Nieuwelaar, and F.W. Vaandrager. Model checker aided design of a controller for a wafer scanner. In *Accepted for the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004)*, October 2004.

[9] M. Hendriks, N.J.M. van den Nieuwelaar, and F.W. Vaandrager. Recognizing finite repetitive scheduling patterns in manufacturing systems. In G. Kendall, E. Burke, and S. Petrovic, editors, *Multidisciplinary International Conference on Scheduling : Theory and Applications(MISTA'03)*, pages 291–319. ASAP, University of Nottingham, United Kingdom, August 2003.

[10] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley-Interscience, Chichester, 1985.

[11] R.M. Milner. Calculus of communicating systems. *Lecture notes in Computer Science*, 92, 1980.

[12] N.J.M. van den Nieuwelaar, J.M. van de Mortel-Fronczak, N.C.W.M. Braspenning, and J.E. Rooda. Predictive scheduling in complex manufacturing machines: scheduling alternatives and algorithm. *submitted to IEEE TAC.*

[13] N.J.M. van den Nieuwelaar, J.M. van de Mortel-Fronczak, N.C.W.M. Braspenning, and J.E. Rooda. Predictive scheduling in complex manufacturing machines: machine-specific constraints. *submitted to IEEE TSM.*

[14] N.J.M. van den Nieuwelaar, J.M. van de Mortel-Fronczak, and J.E. Rooda. Design of supervisory machine control. In *European Control Conference 2003*, 2003.

[15] 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*, 46(10):1572–1583, 2001.

[16] S.E. Ramaswamy and S.B. Joshi. Deadlock-free schedules for automated manufacturing workstations. *IEEE Transactions on Robotics and Automation*, 12(3):391–400, June 1996.

[17] S.A. Reveliotis, M. Lawley, and P. Ferreira. Polynomial-complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Transactions on Automatic Control*, 42(10):1344–1357, 1997.

[18] E. Roszkowska. Supervisory control for deadlock avoidance in compound processes. *IEEE Transactions on Systems, Man, and Cybernetics - part A: Systems and Humans*, 34(1):52–64, January 2004.

[19] N. Viswanadham, Y. Narahari, and T.L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE Transactions on Robotics and Automation*, 6(6):713–723, December 1990.

[20] M. Wennink. *Algorithmic Support for Automated Planning Boards*. PhD thesis, Eindhoven University of Technology, the Netherlands, 1995.

[21] R.A. Wysk, N.S. Yang, and S. Joshi. Detection of deadlocks in flexible manufacturing cells. *IEEE Transactions on Robotics and Automation*, 7(6):853–859, December 1991.

[22] H.J. Yoon and D.Y. Lee. Deadlock-free scheduling of photolithography equipment in semiconductor fabrication. *IEEE Transactions on Semiconductor Manufacturing*, 17(1):42–54, February 2004.