Exploiting Specification Modularity to Prune the Optimization-Space of Manufacturing Systems

João Bastos
Eindhoven University of Technology
Eindhoven, The Netherlands
j.p.nogueira.bastos@tue.nl

Sander Stuijk
Eindhoven University of Technology
Eindhoven, The Netherlands
s.stuijk@tue.nl

Jeroen Voeten
Eindhoven University of Technology
Embedded Systems Innovation
Eindhoven, The Netherlands
j.p.m.voeten@tue.nl

Ramon Schiffelers
Eindhoven University of Technology
Eindhoven, The Netherlands
ASML
Veldhoven, The Netherlands
ramon.schiffelers@asml.com

Henk Corporaal
Eindhoven University of Technology
Eindhoven, The Netherlands
h.corporaal@tue.nl

ABSTRACT

In this paper we address the makespan optimization of industrial-sized manufacturing systems. We introduce a framework which specifies functional system requirements in a compositional way and automatically computes makespan optimal solutions respecting these requirements. We show the optimization problem to be NP-Hard. To scale towards systems of industrial complexity, we propose a novel approach based on a subclass of compositional requirements which we call constraints. We prove that these constraints always prune the worst-case optimization-space thereby increasing the odds of finding an optimal solution (with respect to the additional constraints). We demonstrate the applicability of the framework on an industrial-sized manufacturing system.

CCS CONCEPTS

• **Theory of computation → Formalisms**; • **Computing methodologies → Model development and analysis**; • **Software and its engineering → System modeling languages**; **Specification languages**; *Design languages*; • **Computer systems organization →** *Embedded and cyber-physical systems*;

KEYWORDS

Manufacturing systems, Design exploration, Modular system specification, Formal models, Makespan optimization

ACM Reference Format:
João Bastos, Sander Stuijk, Jeroen Voeten, Ramon Schiffelers, and Henk Corporaal. 2018. Exploiting Specification Modularity to Prune the Optimization-Space of Manufacturing Systems. In *SCOPES '18: 21st International Workshop on Software and Compilers for Embedded Systems, May 28–30, 2018, Sankt Goar, Germany.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3207719.3207728

1 INTRODUCTION

Manufacturing systems are Cyber-Physical Systems which perform operations on batches of products. Production and transport units (the physical part) carry out operations, while a controller (the cyber part) coordinates the order and timing of execution. Examples of these systems include lithography machines or industrial production printers. During their design, functional requirements (such as keeping the order of products in a batch or avoiding collisions) must be satisfied while at the same time demanding performance requirements must be met, for instance regarding productivity.

In [2, 14] a framework is proposed for the specification of both functional and timing requirements for such manufacturing systems. Compared to other frameworks, its novelty lies in the separation of concerns between specification and optimization, together with its compositional support for requirements specification. However, no complexity study is provided with respect to the optimization problem. In this paper we restrict this framework of [14] to batch-oriented manufacturing systems and prove that their optimization is NP-Hard. As a consequence, optimal solutions might take prohibitively long depending on the size of the optimization-space induced by the optimization problem.

When considering real industrial applications, the size of the optimization-space cannot be disregarded [15]. To still support optimization, heuristic approaches can be applied, however, often at the cost of sub-optimal solutions and uncertainty regarding their quality. In this work we propose a novel approach where we further exploit the modularity of the framework as an alternative to heuristic solutions. We introduce classes of compositional requirements, which we call constraints, and show that they effectively prune the (worst-case) optimization-space. Furthermore, this approach still guarantees that optimal solutions can be found when the additional constraints are taken into account.

The approach is inspired by common practices in an industrial setting, where manufacturing systems are typically over-specified

[11] and in which over-specification is used implicitly and unconsciously to deal with complexity. Examples of over-specification that we have encountered in industrial cases are for instance: disallowing multiple mapping possibilities for an operation or enforcing the static ordering of system operations.

Our approach allows system designers to deal consciously with over-specification by an explicit formalization in terms of constraints. This constraint-oriented approach provides control over the worst-case size of the optimization-space of the system. Furthermore, it is guaranteed that functionally correct controllers are obtained.

Contribution: Our contributions are summarized below:

- We refine the framework of [14] to deal with batch-oriented systems and exploit the concept of constraints as a formal means to deal with optimization complexity and prove that these constraints effectively prune the (worst-case) optimization space.
- We show that the framework scales to an industrial-sized manufacturing system.

Overview: The remainder of this paper is organized as follows. Section 2 summarizes the framework which we refine. The batch optimization problem is explained in Section 3 and proven to be NP-Hard in Section 4. Section 5 elaborates on the modular specification of batch-oriented logistics. Section 6 shows that constraining effectively prunes the (worst-case) optimization-space. Section 7 exemplifies the approach with a real industrial case-study. Section 8 discusses related work and Section 9 concludes.

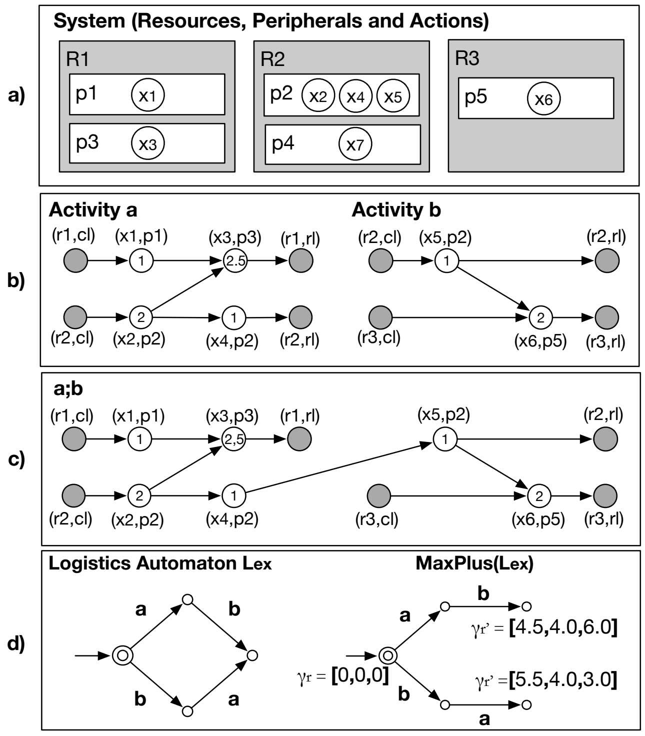

Figure 1: Overview of the specification framework [14].

2 SPECIFICATION FRAMEWORK

In this paper, we assume as a starting point the framework of [14]. In this section, we summarize the building blocks of this framework and its essential properties.

A manufacturing system is decomposed into a set of *peripherals* ($\mathcal{P}$), *actions* ($\mathcal{A}$) and *resources* ($\mathcal{R}$). Each peripheral can execute *actions.* An action describes an atomic behavior of the system, e.g. the movement of a motor or the actuation of an on/off peripheral such as a clamp. The complete set of actions describes all behavior that the system can exhibit. Peripherals are aggregated into *resources*, which can be *claimed* and *released.* Figure 1 a) depicts a synthetic example of a system decomposed into resources R1, R2 and R3, peripherals p1, p2, p3, p4 and p5 and actions $x1$, $x2$, $x3$, $x4$, $x5$, $x6$ ands $x7$. Using peripheral actions, deterministic functional behaviors of the system can be constructed as Activities. An *activity* is a Directed Acyclic Graph (DAG), consisting of a set N of nodes and a set $\rightarrow$ of dependencies between nodes. Nodes refer to either an action executed by a peripheral (associated with a pair $(x, p) : x \in \mathcal{A}$ and $p \in \mathcal{P}$), or a claim (cl) or release (rl) of a resource (associated with a pair $(r, v) : r \in \mathcal{R}$ and $v \in \{cl, rl\}$). Figure 1 b) depicts two such activities, a and b.

Multiple activities can be sequenced to form a new activity. Resource sharing and concurrency between multiple activities is taken into account by the correct claiming and releasing of resources between activities. Figure 1 c) depicts the result of sequencing activities a and b. This can be formally achieved by applying the semi-colon (;) operator [14], yielding a;b. Intuitively, the releasing and claiming of shared resources must be correctly matched and replaced by a new dependency such that the resulting DAG is itself an activity.

Besides functional specification, the framework requires a temporal specification for performance optimization. A function $T : \mathcal{A} \rightarrow \mathbb{R}_{\geq 0}$ maps each action to its fixed execution time. Consequently, given an activity, the execution times can be lifted to the level of nodes, where the execution times of claim and releases nodes are assumed to be 0. In Figures 1 b) and c) the timing information is annotated within the nodes (except for the release and claim nodes). Since actions are executed on resources, a *resource time stamp* vector $\gamma_R : \mathcal{R} \rightarrow \mathbb{R}^{-\infty}$ represents the system state in terms of resource availability. For each $r \in \mathcal{R}$, $\gamma_R(r) \in \mathbb{R}^{-\infty}$ is the availability time of resource r. We use $\mathbf{0}_R$ to denote the resource time-stamp with all zero-value entries.

Starting from resource vector γ_R the execution of an activity a leaves the system in a new state γ'_R. This new state is determined by computing the completion times of each node in the activity, taken into account the dependencies and execution times of its nodes. For this purpose, $(max, +)$ algebra [1] is used. In [14] it is shown that $\gamma'_R = M_a \otimes \gamma_R$, where M_a is the $(max, +)$ matrix of activity a and where $\otimes$ is the (max,+) matrix multiplication operator. The new system state γ'_R determines the makespan of the activity execution, represented by $mks(a, \gamma_R)$. The makespan is given by the largest value in γ'_R which is by definition the $(max, +)$ norm $\parallel \gamma'_R \parallel$. Notice that the makespan of the execution of a sequence $a_1 a_2 \cdots a_n$ of activities (starting from γ_R) is given by $M_{(a_1;a_2;\cdots;a_n)} \otimes \gamma_R$ (where ; denotes the activity sequencing operator). In [14] it is shown that this expression is equivalent to $M_{a_1} \otimes M_{a_2} \otimes \cdots \otimes M_{a_n} \otimes \gamma_R$.

3 BATCH MAKESPAN OPTIMIZATION

Having described the basic framework for the specification of manufacturing systems, we now introduce the batch optimization problem. For this purpose, we start by defining the necessary concepts for the specification of the logistics of batch-oriented manufacturing systems. A sequence of activities can model the *complete* manufacturing of a product, or batch of products, where a single activity models one manufacturing operation. The collection of all allowed activity sequences of a system defines the *logistics* behavior. Such a collection is encoded in a *logistics automaton.*

Definition 3.1 (Logistics automaton). A logistics automaton is a tuple $\langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$, where $\mathcal{S}$ is a finite (possibly empty) set of states, $\mathcal{A}ct$ is a finite (possibly empty) set of activities, $\dot{\rightarrow} \subseteq \mathcal{S} \times \mathcal{A}ct \times \mathcal{S}$ is a transition relation, and $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states, where $\mathcal{S}_0 = \emptyset$ if $\mathcal{S} = \emptyset$ and $\mathcal{S}_0 = \{s_0\}$ otherwise. Let $s \xrightarrow{a} s'$ be a shorthand for $(s, a, s') \in \dot{\rightarrow}$. The following additional properties must hold:

- Acyclicity: there exists no $s \in \mathcal{S}$ such that $s \dot{\rightarrow}^+ s$, where $\dot{\rightarrow}^+$ is the transitive closure of $\dot{\rightarrow}$, and where $s \dot{\rightarrow} t$ denotes that $s \xrightarrow{a} t$ for some $a \in \mathcal{A}ct$;
- Reachability: if $\mathcal{S} \neq \emptyset$ then for all $s \in \mathcal{S}$, $s_0 \dot{\rightarrow}^* s$, where $\dot{\rightarrow}^*$ is the reflexive transitive closure of $\dot{\rightarrow}$.

Definition 3.2 (Language of a logistics automaton). Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a logistics automaton. The language $\mathcal{L}(L)$ of L is defined by

$$\mathcal{L}(L) = \begin{cases} \emptyset, & \text{if } \mathcal{S}_0 = \emptyset \\ \{\underline{a} \in \mathcal{A}ct^* \mid s_0 \xrightarrow{\underline{a}} s \text{ for some } s \in \mathcal{S} \\ \text{and } \not\dot{\rightarrow}\}, & \text{if } \mathcal{S}_0 = \{s_0\} \end{cases}$$

Here $\mathcal{A}ct^*$ denotes the collection of all sequences of activities in $\mathcal{A}ct$. Each $\underline{a} \in \mathcal{A}ct^*$ is of the form $a_1...a_n$, where $a_i \in \mathcal{A}ct$ $(1 \leq i \leq n)$. For $n = 0$, $\underline{a}$ is the empty activity sequence denoted by ϵ. For states $s, s' \in \mathcal{S}$ and $\underline{a} = a_1, ..., a_n \in \mathcal{A}ct^*$ we let $s \xrightarrow{\underline{a}} s'$ denote the existence of $s_1, \cdots, s_n \in \mathcal{S}$ such that $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n = s'$. Further $s \xrightarrow{a}$ denotes that $s \xrightarrow{a} s'$ for some $s' \in \mathcal{S}$, $s \dot{\rightarrow}$ denotes that $s \xrightarrow{a} s'$ for some $a \in \mathcal{A}ct$ and $s' \in \mathcal{S}$ and $s \not\dot{\rightarrow}$ denotes that $s \dot{\rightarrow}$ does not hold. Note that $\mathcal{L}(L) = \emptyset$ if $\mathcal{S} = \emptyset$ and $\mathcal{L}(L) = \{\epsilon\}$ if $\mathcal{S} = \{s_0\}$. Notice also that any sequence in the language should "run to completion", i.e. finish in a state with no outgoing transitions. As a consequence languages of logistics automata are not prefix closed (except when the language equals $\{\epsilon\}$).

Figure 1 d) depicts an example logistics automaton L_{ex}. Two activities, a and b are used to specify different allowed activity sequences. The activities are annotated on the edges. Each activity sequence is determined by a path starting from the initial state (depicted with an extra circumference and no incoming edges) and finishing on a final state (a state without outgoing edges). In this way the automaton encodes a set of activity sequences, where each sequence represents one possible way to successfully manufacture a product, or batch of products. Hence the logistics automaton encodes a *language* of allowed activity sequences. For example, L_{ex} encodes activity sequences ab and ba and thus $\mathcal{L}(L_{\text{ex}}) = \{ab, ba\}$. Each activity sequence can result in different manufacturing completion times when taking into account the temporal execution of its activities. For example, the completion times of sequences ab and ba are 6 and 5.5 respectively . These completion times are obtained as the makespan of the corresponding activity sequences (mks(a;b, $\mathbf{0}_R$) and mks(b;a, $\mathbf{0}_R$)). Thus the Batch Makespan Optimization (BMO) involves determining the activity sequence in the language of the logistics automaton with minimal makespan.

Problem 1 (Batch Makespan Optimization). *Given a Logistics automaton L determine* $\underline{a} \in \mathcal{L}(L)$ *such that*

$$mks(\underline{a}, \mathbf{0}_R) \leq mks(\underline{a}', \mathbf{0}_R)$$

for all $\underline{a}' \in \mathcal{L}(L)$. *Here we let* $mks(\underline{a}, \mathbf{0}_R)$ *denote* $mks(a_1; ...; a_n, \mathbf{0}_R)$ *for brevity.*

This problem can be solved in two steps. The first is to build a timed expansion of logistics automaton L using the (max,+) characterization of its activities as a new automaton which we call the (max,+) automaton denoted by MaxPlus(L). It can be shown that MaxPlus(L) satisfies Definition 3.1 and is thus a logistics automaton itself.

Definition 3.3 ((max,+) automaton). Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a logistics automaton. First define MaxPlusStates(L) as the smallest set V satisfying inference rules (1) and (2):

$$\frac{\mathcal{S}_0 = \{s_0\}}{(s_0, \mathbf{0}_R) \in V} \ (1) \qquad \frac{(s, \gamma_R) \in V \quad s \xrightarrow{a} s'}{(s', \gamma_R \otimes M_a) \in V} \ (2)$$

Here γ_R denotes a resource time-stamp vector and $\mathbf{0}_R$ denotes the resource time-stamp vector containing only 0 valued entries. M_a denotes the $(max, +)$ matrix corresponding to activity $a \in \mathcal{A}ct$ and $s, s' \in \mathcal{S}$. Then we define MaxPlus(L) as

$$(\text{MaxPlusStates}(L), \mathcal{A}ct, \dot{\rightarrow}', \mathcal{S}_0')$$

where $\rightarrow' = \left\{(s, \gamma_R), a, (s', \gamma_R') \in \text{MaxPlusStates}(L) \times \mathcal{A}ct \times \text{MaxPlusStates}(L) \mid s \xrightarrow{a} s' \text{ and } \gamma_R' = \gamma_R \otimes M_a\right\}$ and $\mathcal{S}_0' = \emptyset$ if $\mathcal{S}_0 = \emptyset$ and $\mathcal{S}_0' = \{(s_0, \mathbf{0}_R)\}$ otherwise.

As an example consider the (max,+) automaton of logistics automaton L_{ex}, MaxPlus(L_{ex}), depicted in Figure 1 d). The temporal execution of activity sequence ab and ba leads to different resource time-stamp vectors and thus the final state of L_{ex} is duplicated in MaxPlus(L_{ex}). Note that the state-space induced by MaxPlus(L) corresponds to the optimization-space of the BMO problem. Therefore, the second step is to explore MaxPlus(L) to find $\underline{a} \in \mathcal{L}(\text{MaxPlus}(L))$ for which the makespan is minimal. This is achieved by finding a final state for which the $(max, +)$ norm of the resource time-stamp vector is minimal. An activity sequence $\underline{a}$ terminating in such a state is a solution to the BMO problem.

4 COMPLEXITY ANALYSIS

To show that BMO is NP-Hard, we first show that its decision version, the Batch Makespan Satisfaction (BMS) problem (whether an activity sequence exists with makespan below a given bound), is NP-Complete. It is well-known that the Weighted Set Partitioning problem (WSP) problem [**?**] is NP-Complete [6]. Here a set of n

elements with integer valued sizes, for which the total size is equal to S, can be partitioned into two subsets of equal size $S/2$. WSP can be reduced to our BMS problem. We define two resources R_1 and R_2 (each with one peripheral p_1 and p_2). For each combination of element and resource we defined an activity, $a^{r_1}_{x_i} : (R_1, cl) \rightarrow (x_i, p_1) \rightarrow (R_1, rl)$ and $a^{r_2}_{x_i} : (R_2, cl) \rightarrow (x_i, p_2) \rightarrow (R_2, rl)$), where x_i refers to element i (with $1 \leq i \leq n$) and where the execution time of x_i corresponds to the size of the ith element. We further define a logistics automaton with states $\mathcal{S}$, labeled $s_0, ..., s_n$. The transitions are defined as: $s_i \xrightarrow{a^{r_1}_{x_i}} s_{i+1}$ and $s_i \xrightarrow{a^{r_2}_{x_i}} s_{i+1}$ for $0 \leq i \leq n$. Such that all combinations of possible partitions are included in the automaton as activity sequences. Therefore, a partitioning is possible iff an activity sequence exists with a makespan equal to $S/2$. Thus, we have reduced a known NP-Complete problem to the BMS problem. Further, it is not difficult to see that a possible solution can be verified in polynomial time. Hence, the BMS problem is NP-Complete. The BMO problem is NP-Hard, since the BMS problem can be solved in polynomial time using a solution of the BMO problem. For assume we have an optimal solution $\underline{a_o}$ for the BMO problem. If $mks(\underline{a_o}, \mathbf{0}_R)$ is less than a given bound B, a solution with a lower makespan than B exists and therefore the answer to BMS problem is positive. Otherwise, no solution exists with makespan lower than B, yielding a negative answer to the BMS problem. Thus, BMO is NP-Hard.

5 MODULAR LOGISTICS SPECIFICATION

Even though a logistics automaton is able to encode the complete manufacturing of a batch of products, for large batch sizes or complex manufacturing jobs a monolithic automaton is not desired. A modular and compositional methodology is more suitable to deal with specification complexity, maintainability and understandability. Therefore, in this work we restrict our framework [14] towards batch-oriented systems and take inspiration from the constraint-oriented specification style of the LOTOS framework [3] and the compositional specification of requirements in CIF [13] by defining a composition operator on logistics automata. The operator allows batches to be specified individually (i.e. each product flow can be specified by an individual automaton) and then composed to obtain an automata that encodes all the logistics requirements for the batch of products.

Definition 5.1 (Composition of Logistics Automata). Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ be logistics automata. Before we define the composition automaton $L_1 \odot L_2$, we first define relation $\dot{\rightarrow} \subseteq \mathcal{S}_1 \times (\mathcal{A}ct_1 \cup \mathcal{A}ct_2) \times \mathcal{S}_2$ as the smallest set V satisfying the following inference rules:

$$\frac{s \xrightarrow{a}_1 s' \quad a \in \mathcal{A}ct_1 \backslash \mathcal{A}ct_2}{(s,t) \xrightarrow{a} (s',t)} \quad (1)$$

$$\frac{s \xrightarrow{a}_1 s' \quad t \xrightarrow{a}_2 t' \quad a \in \mathcal{A}ct_1 \cap \mathcal{A}ct_2}{(s,t) \xrightarrow{a} (s',t')} \quad (2)$$

$$\frac{t \xrightarrow{a}_2 t' \quad a \in \mathcal{A}ct_2 \backslash \mathcal{A}ct_1}{(s,t) \xrightarrow{a} (s,t')} \quad (3)$$

where $s, s' \in \mathcal{S}_1$ and $t, t' \in \mathcal{S}_2$. Now define the set of states $\mathcal{S}$ of the composition automaton as

$$\mathcal{S} = \begin{cases} \emptyset, & \text{if } \mathcal{S}_1 = \emptyset \text{ or } \mathcal{S}_2 = \emptyset \\ \{(s,t) \in \mathcal{S}_1 \times \mathcal{S}_2 \mid (s_{0_1}, s_{0_2}) \dot{\rightarrow}^* (s,t) & \\ \text{ and for some } (s',t') \in \mathcal{S}_1 \times \mathcal{S}_2 \text{ with} & \\ s' \not\dot{\rightarrow}_1 \text{ and } t' \not\dot{\rightarrow}_2, (s,t) \dot{\rightarrow}^* (s',t')\} & \text{if } \mathcal{S}_{0_1} = \{s_{0_1}\} \\ & \text{ and } \mathcal{S}_{0_2} = \{s_{0_2}\} \end{cases}$$

Further define $\dot{\rightarrow}' = \{((s,t), a, (s',t')) \subseteq \dot{\rightarrow} \mid (s,t), (s',t') \in \mathcal{S}\}$ and

$$\mathcal{S}_0 = \begin{cases} \emptyset, & if \mathcal{S} = \emptyset \\ \{s_{0_1}, s_{0_2}\} & otherwise \end{cases}$$

Finally, the composition automaton $L_1 \odot L_2$ is defined as:

$$\langle \mathcal{S}, \mathcal{A}ct_1 \cup \mathcal{A}ct_2, \dot{\rightarrow}', \mathcal{S}_0 \rangle$$

As an example consider logistics automata L_1 and L_2 depicted in Figure 2 a) and b). Each automata models the manufacturing life-cycle of single product. We can compose them into a new logistics automaton that reflects a two-product system using the composition operator $\odot$. The resulting composition $L_1 \odot L_2$ is shown in Figure 2 c). One complete activity sequence from the start state to a final state in $L_1 \odot L_2$ represents the full execution of both product life-cycles. In fact $L_1 \odot L_2$ captures all the allowed interleavings of activities satisfying the life-cycles of both L_1 and L_2. In this fashion, batches of products can be specified, including batches with different product types and corresponding life-cycles (an important aspect of flexible manufacturing systems [9]).

Using logistics automata we capture the activity flow of a product and ensure the completion of the manufacturing of this product. In addition we can express constraints within the same product, concerning for instance a safe handover between resources. For a complete specification of the manufacturing of a batch of products, we also need to specify constraints across different product flows. We will express such constraints in terms of *constraint automata* and introduce a *constraint operator* to compose them with logistics automata. Constraint automata capture constraints between different products such as ordering constraints (e.g. FIFO ordering for a batch), safety constraints (e.g. access to exclusive safety areas), resource capacity constraints (e.g. a resource must be empty before receiving a product) or other constraints that are expressed as dependencies across different product flows.

Definition 5.2 (Constraint automaton). A constraint automaton is a tuple $\langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$, where $\mathcal{S}$ is a finite (possibly empty) set of states, $\mathcal{A}ct$ is a finite (possibly empty) set of activities, $\dot{\rightarrow} \subseteq \mathcal{S} \times \mathcal{A}ct \times \mathcal{S}$ is a transition relation, and $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states, where $\mathcal{S}_0 = \emptyset$ if $\mathcal{S} = \emptyset$ and $\mathcal{S}_0 = \{s_0\}$ otherwise. The following additional property must hold:

- Reachability: if $\mathcal{S} \neq \emptyset$ then for all $s \in \mathcal{S}, s_0 \dot{\rightarrow}^* s$.

A constraint automaton encodes a language, just as a logistics automaton does.

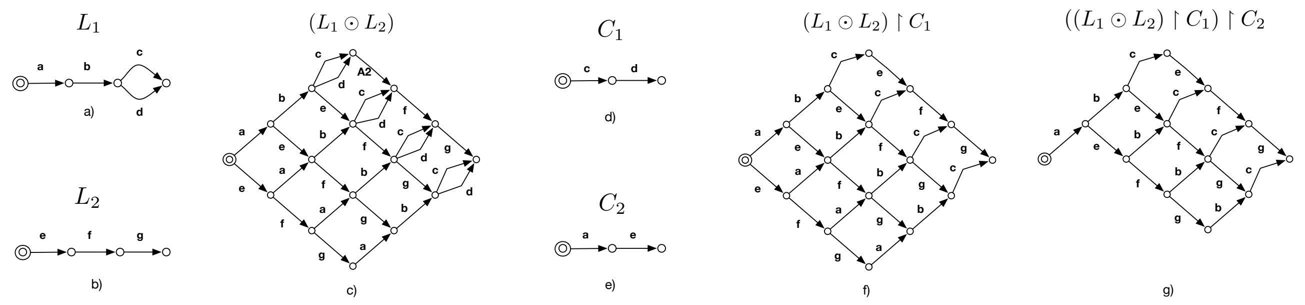
L_1
$(L_1 \odot L_2)$
C_1
$(L_1 \odot L_2) \upharpoonright C_1$
$((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_2$
L_2
C_2
A2
a)
b)
c)
d)
e)
f)
g)

Figure 2: Example logistics automata L_1 a) and L_2 b); c) the resulting logistics automaton $L_1 \odot L_2$; example constraint automata C_1 d) and C_2 e); f) the resulting logistics automaton $(L_1 \odot L_2) \upharpoonright C_1$ and g) the resulting logistics automaton $((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_2$.

Definition 5.3 (Language of a constraint automaton). Let $C = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a constraint automaton. The language $\mathcal{L}(C)$ of C is defined by

$$\mathcal{L}(C) = \begin{cases} \emptyset, & \text{if } \mathcal{S}_0 = \emptyset \\ \{\underline{a} \in \mathcal{A}ct^* \mid s_0 \xrightarrow{\underline{a}} s \text{ for some } s \in \mathcal{S}\}, & \text{if } \mathcal{S}_0 = \{s_0\} \end{cases}$$

Constraints can be applied to logistics automata through the constraint operator. A constraint automaton $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ is called *a constraint on* logistics automaton $L = \langle \mathcal{S}_1, \mathcal{A}ct_1\ \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ if $\mathcal{A}ct_2 \subseteq \mathcal{A}ct_1$. Applying a constraint C to automaton L yields a new logistics automaton which is denoted by $L \upharpoonright C$.

Definition 5.4 (Constraint Operator). Let $L = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ be a logistics automaton let $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ be a constraint on L (so that $\mathcal{A}ct_2 \subseteq \mathcal{A}ct_1$). Before we define $L \upharpoonright C$ we first define relation $\dot{\rightarrow} \subseteq \mathcal{S}_1 \times \mathcal{A}ct_l \times \mathcal{S}_2$ as the smallest set V satisfying the following inference rules:

$$\frac{s \xrightarrow{a}_1 s' \quad a \in \mathcal{A}ct_1 \backslash \mathcal{A}ct_2}{(s,t) \xrightarrow{a} (s',t)} \quad (1)$$

$$\frac{s \xrightarrow{a}_1 s' \quad t \xrightarrow{a}_2 t' \quad a \in \mathcal{A}ct_1 \cap \mathcal{A}ct_2}{(s,t) \xrightarrow{a} (s',t')} \quad (2)$$

where $s, s' \in \mathcal{S}_l$ and $t, t' \in \mathcal{S}_c$. Now define the set of states $\mathcal{S}$ of the constrained automaton as

$$\mathcal{S} = \begin{cases} \emptyset, & \text{if } \mathcal{S}_1 = \emptyset \text{ or } \mathcal{S}_2 = \emptyset \\ \{(s,t) \in \mathcal{S}_1 \times \mathcal{S}_2 \mid (s_{0_1}, s_{0_2}) \dot{\rightarrow}^* (s,t) & \\ \text{ and for some } (s',t') \in \mathcal{S}_1 \times \mathcal{S}_2 \text{ with} & \\ s' \not\rightarrow_1, (s,t) \dot{\rightarrow}^* (s',t')\} & \text{if } \mathcal{S}_{0_1} = \{s_{0_1}\} \\ & \text{and } \mathcal{S}_{0_2} = \{s_{0_2}\} \end{cases}$$

Further define $\dot{\rightarrow}' = \{((s,t), a, (s',t')) \subseteq \dot{\rightarrow} \mid (s,t), (s',t') \in \mathcal{S}\}$ and

$$\mathcal{S}_0 = \begin{cases} \emptyset, & \text{if } \mathcal{S} = \emptyset \\ \{s_{0_1}, s_{0_2}\} & \textit{otherwise} \end{cases}$$

Finally, the constrained automaton $L \upharpoonright C$ is defined as:

$$\langle \mathcal{S}, \mathcal{A}ct_1, \dot{\rightarrow}', \mathcal{S}_0 \rangle$$

As an example consider automata C_1 and C_2 in Figures 2 d) and e). C_1 models a constraint on the order of activities c and d of the product modeled by L_1, and C_2 a constraint on the input order of both products by ordering activities a and f of L_1 and L_2. The result of the constraint operation is depicted in Figures 2 f) showing $(L_1 \odot L_2) \upharpoonright C_1$ and g) showing $((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_2$.

In logistics automaton $(L \upharpoonright C)$, C constraints the behavior of L resulting in a subset of the original language. This is claimed by the following Lemma:

Lemma 5.5 (Language constraining). *Let L be a logistics automaton and let C be a constraint on L. Then $\mathcal{L}(L \upharpoonright C) \subseteq \mathcal{L}(L)$.*

Note that the constraint operator requires the logistics automaton to run to completion, while this is not true for the constraint automaton. In other words constraint automata capture only safety requirements (expressing that nothing bad should happen) while logistics automata capture both safety requirements and liveness requirements (expressing that something good happens eventually, namely the completion of the different products in the batch). On the contrary, the composition operator requires that both logistics automata run to completion. It was not strictly necessary to add the concepts of constraint automata and constraint operator since constraints can in principle be encoded as logistics automata (by unfolding recursive loops sufficiently often). However, the automaton obtained by composing would in many cases have a larger state-space than the constrained automaton, which we try to prevent. In addition logistics automata encoding constraints would be incomprehensible because of their size and also less reusable. For these reasons we decided to introduce new concepts in the form of constraint automata and the constraint operator.

6 OPTIMIZATION-SPACE PRUNING

By observing Figure 2 one may conclude that constraining leads to state-space reduction. Indeed the size of $(L_1 \odot L_2) \upharpoonright C_1$ is smaller than that of $L_1 \odot L_2$ and the size of $((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_2$ is smaller than that of $(L_1 \odot L_2) \upharpoonright C_1$. One can even show that the same holds for the (max,+) optimization-spaces of these automata. This observation led to the main idea behind this work of exploiting constraints to prune the optimization-space. However, even though constraining leads to a reduction of the encoded language (see Lemma 5.5) it

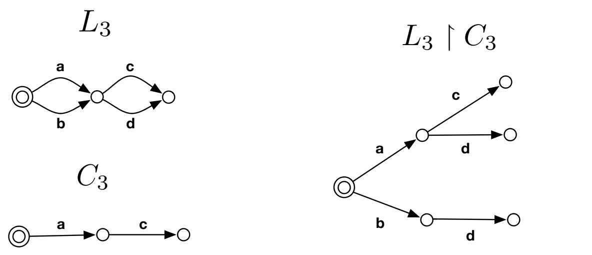

Figure 3: $L_3 \upharpoonright C_3$ **results in a larger state-space than** L_3**.**

does not necessarily imply a reduction of the state-space, neither of the logistics automaton nor of the (max,+) optimization-space. For instance, consider logistics automata L_3 and constraint automata C_3 and the constrained logistics automaton $L_3 \upharpoonright C_3$ depicted in Figure 3. Here the size of $L_3 \upharpoonright C_3$ is larger than that of L_3.

In this section we establish sufficient conditions for the constraint operation to reduce the *worst-case* optimization-space (which is one of the main contributions of this paper). To this end we introduce a relation $\sqsubseteq$ on logistics automata which is stronger than language inclusion (i.e. $L_1 \sqsubseteq L_2$ implies $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ and further satisfies the property that $L_1 \sqsubseteq L_2$ implies that the size of the state-space of L_1 never exceeds that of L_2). We further define the worst-case optimization-space of logistics automaton L as a new logistics automaton Tree(L) and prove that MaxPlus(L) $\sqsubseteq$ Tree(L) and that indeed Tree($L \upharpoonright C$) $\sqsubseteq$ Tree(L) for any L and deterministic constraint automata C. This implies that the constraining with a deterministic constraint leads to a reduction of the worst-case state-space.

Definition 6.1 (Inclusion). Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ be logistics automata. Then L_1 is included in L_2, written $L_1 \sqsubseteq L_2$, if and only if $\mathcal{A}ct_1 = \mathcal{A}ct_2$ and either i) $\mathcal{S}_1 = \emptyset$ or ii) $\mathcal{S}_{0_1} = \{s_{0_1}\}$ and $\mathcal{S}_{0_2} = \{s_{0_2}\}$ and there exists an injective relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ satisfying:

(1) $(s_{0_1}, s_{0_2}) \in R$;
(2) For all $(s_1, s_2) \in R$ and $a \in \mathcal{A}ct_1$ if $s_1 \xrightarrow{a}_1 s_1'$ (for some $s_1' \in \mathcal{S}_1$) then $s_2 \xrightarrow{a}_2 s_2'$ (for some $s_2' \in \mathcal{S}_2$) and $(s_1', s_2') \in R$;
(3) For all $(s_1, s_2) \in R$ if $s_2 \dot{\rightarrow}_2$ then $s_1 \dot{\rightarrow}_1$.

It is not hard to prove that if a logistics automaton is included in another logistics automaton, the size of the state-space of the former never exceeds that of the latter. As an example consider the automata of Figure 2 c) and f) and note that $(L_1 \odot L_2) \upharpoonright C_1 \sqsubseteq (L_1 \odot L_2)$ and that the state-space of $(L_1 \odot L_2) \upharpoonright C_1$ does not exceed that of $L_1 \odot L_2$. Considering the automata of Figure 3, on the other hand, we observe that in this case $L_3 \upharpoonright C_3 \not\sqsubseteq L_3$ and that the size of the state-space of $L_3 \upharpoonright C_3$ is larger than that of L_3.

Definition 6.2 (Tree automaton). Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a logistics automaton. We first define Paths(L) as the smallest set V satisfying:

$$\frac{\mathcal{S}_0 = \{s_0\}}{s_0 \in V}\ (1) \qquad \frac{q\ s \in V \quad s \xrightarrow{a} s'}{q\ s\ a\ s' \in V}\ (2)$$

Here $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}ct$. *Paths*(L) contains sequences of elements in $\mathcal{S}$ and $\mathcal{A}ct$. Each sequence is of the form $s_0\ a_0\ s_1\ a_1\ \cdots\ s_n$ and encodes the path from starting state s_0 to state s_n (via transitions labeled with activities $a_0 \cdots a_{n-1}$ and intermediate states $s_1 \cdots s_{n-1}$). In inference rule (2), $q\ s$ refers to a path that ends in state s. Note that q can refer to an empty sequence of elements. Remark that Paths(L) $= \emptyset$ if $\mathcal{S} = \emptyset$. We now define Tree(L) as

$$(\text{Paths}(L), \mathcal{A}ct, \dot{\rightarrow}', \mathcal{S}_0')$$

where $\dot{\rightarrow}' = \{(q\ s, a, q\ s\ a\ s') \in \text{Paths}(L) \times \mathcal{A}ct \times \text{Paths}(L) \mid s \xrightarrow{a} s'\}$ and $\mathcal{S}_0' = \emptyset$ if $\mathcal{S}_0 = \emptyset$ and $\mathcal{S}_0' = \{s_0\}$ otherwise. Note that a Tree automaton is a logistic automaton.

Lemma 6.3. *Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ and $\mathcal{T}(L)$ be a logistic automaton. Then MaxPlus(L) $\sqsubseteq$ Tree(L).*

Proof. The results follows directly in case $\mathcal{S} = \emptyset$. Otherwise $S_0 = \{s_0\}$ for which case we define relation $R \subseteq$ MaxPlusStates(L) $\times$ Paths(L) as the smallest set V satisfying the following inference rules:

$$\frac{\mathcal{S}_0 = \{s_0\}}{((s_0, \mathbf{0}_R), s_0) \in V}\ (1) \quad \frac{((s, \gamma_R), q\ s) \in V \quad s \xrightarrow{a} s'}{((s', \gamma_R \otimes M_a), q\ s\ a\ s') \in V}\ (2)$$

It is not hard to show that R satisfies conditions *1., 2. and 3.* of Definition 6.1. To prove that R is injective we have to show that any two pairs in R with equal right-hand elements also have equal left-hand elements. We will do this by induction on the depth of the derivation tree of one of the pairs. Notice that the pairs are either both derived by inference rule (1) or both derived by inference rule (2). In the first case their left-hand elements are obviously the same. In the other case the pairs must be of the forms $((s', \gamma_R \otimes M_a), q\ s\ a\ s')$ and $((s', \gamma_R' \otimes M_a), q\ s\ a\ s')$ and we further know that $((s, \gamma_R), q\ s) \in R$ and $((s, \gamma_R'), q\ s) \in R$. By induction we then have $\gamma_R = \gamma_R'$. □

Finally, we show that if C is a deterministic constraint automaton, the constraining of a logistics automaton L with C never results in a larger worst-case optimization-space.

Lemma 6.4. *Let $L = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ be a logistics automaton and $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ a constraint on L. If C is deterministic, then Tree($L \upharpoonright C$) $\sqsubseteq$ Tree(L).*

Proof. Let Tree(L) $= \langle \text{Paths}(L), \mathcal{A}ct_1, \dot{\rightarrow}_{\text{Tree}(L)}, S_{0_{\text{Tree}(L)}} \rangle$ and Tree($L \upharpoonright C$) $= \langle \text{Paths}(L \upharpoonright C), \mathcal{A}ct_1, \dot{\rightarrow}_{\text{Tree}(L \upharpoonright C)}, S_{0_{\text{Tree}(L \upharpoonright C)}} \rangle$. Notice that these tree automata have the same alphabets. Now either i) Paths($L \upharpoonright C$) $= \emptyset$ or ii) $S_{0_{\text{Tree}(L \upharpoonright C)}} = (s_0, c_0)$ and $S_{0_{\text{Tree}(L)}} = s_0$ where $s_0 \in \mathcal{S}_{0_1}$ and $c_0 \in \mathcal{S}_{0_2}$. In case i) Tree($L \upharpoonright C$) $\sqsubseteq$ Tree(L) holds by definition. Otherwise, we define $R \subseteq$ Paths($L \upharpoonright C$) $\times$ Paths(L) as the smallest set V satisfying the following inference rules:

$$\frac{S_{0_{\text{Tree}(L \upharpoonright C)}} = \{(s_0, c_0)\}}{((s_0, c_0), s_0) \in V}\ (1)$$

$$\frac{\big(q\ (s, c), p\ s\big) \in V \quad (s, c) \xrightarrow{a}_{L \upharpoonright C} (s', c')}{\big(q\ (s, c)\ a\ (s', c'), p\ s\ a\ s'\big) \in V}\ (2)$$

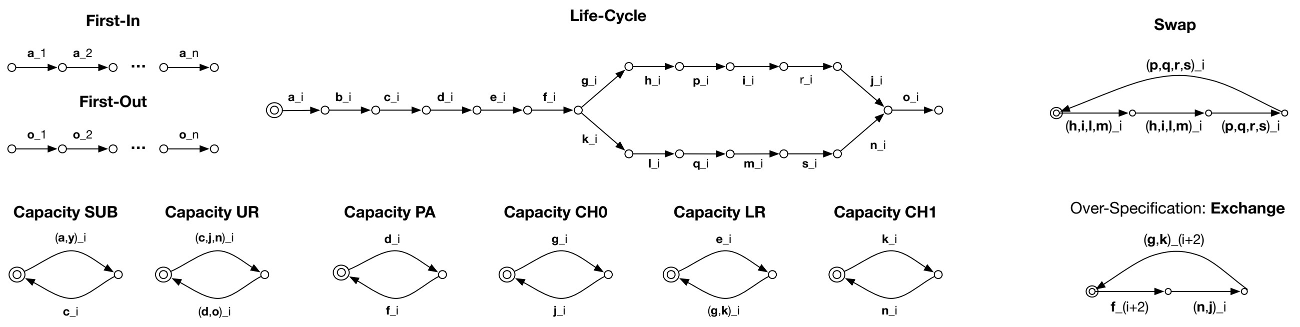

Figure 4: Case study automata: logistics (Life-Cycle) and constraints (Capacity, First-In and First-Out, Swap and Assignment).

Recall that $q\ (s,c)$ and $p\ s$ refer to a paths that end in states $(s,c) \in \mathcal{S}_{L\upharpoonright C}$ and $s \in \mathcal{S}_1$ respectively. We have to show that R is injective and satisfies condition 1., 2. and 3. of Definition 6.1. Now clearly $((s_0,c_0),s_0) \in R$ so condition 1. is satisfied. For condition 2. let $\left(q\ (s,c), p\ s\right) \in R$ and assume $q\ (s,c) \xrightarrow{a}_{\text{Tree}(L\upharpoonright C)} q'$ (for some $a \in \mathcal{A}ct_1$ and $q' \in \text{Paths}(L \upharpoonright C)$). Then q' must be of the form $q\ (s,c)\ a\ (s',c')$ for some $(s',c') \in \mathcal{S}_{L\upharpoonright C}$ for which $(s,c) \xrightarrow{a}_{L\upharpoonright C} (s',c')$. But then $s \xrightarrow{a}_1 s'$ and thus $p\ s \xrightarrow{a}_{\text{Tree}(L)} p\ s\ a\ s'$. Consequently $\left(q\ (s,c)\ a\ (s',c'), p\ s\ a\ s'\right) \in R$, which follows from inference rule (2). For condition 3. let $\left(q\ (s,c), p\ s\right) \in R$ and assume $p\ s \dot{\to}_{\text{Tree}(L)}$. Then $s \dot{\to}_1$ and since $(s,c) \in \mathcal{S}_{L\upharpoonright C}$ also $(s,c) \dot{\to}_{L\upharpoonright C}$. But then also $q\ (s,c) \dot{\to}_{\text{Tree}}\ (L \upharpoonright C)$.

Finally to prove that R injective we have to show that any two pairs in R with equal right-hand elements also have equal left-hand elements. Notice that such two pairs are either both produced by inference rule (1) or both by inference rule (2). In the first case obviously both left-hand sides are equal. In the second case the pairs are of the form $\left(q_1\ (s,c_1)\ a\ (s',c_1'), p\ s\ a\ s'\right)$ and $\left(q_2\ (s,c_2)\ a\ (s',c_2'), p\ s\ a\ s'\right)$. Then $\left(q_1\ (s,c_1), p\ s\right) \in R$ and $\left(q_2\ (s,c_2), p\ s\right) \in R$. By induction we then have $q_1 = q_2$ and $c_1 = c_2$. We further know that $(s,c_1) \xrightarrow{a}_{L\upharpoonright C} (s',c_1')$ and $(s,c_1) \xrightarrow{a}_{L\upharpoonright C} (s',c_2')$. Now since C is deterministic, we also have that $c_1' = c_2'$. □

In conclusion, the addition of any deterministic constraint results in a reduced worst-case optimization space. In addition it results in an upperbound of the optimal makespan corresponding to the unconstrained specification. Constraints therefore allows the conscious use of over-specification (addition of constraints on top of the original system specification) to control the optimization-space and effectively determine upperbounds on the optimal makespan. We show an example of such a realistic case in the following section.

7 CASE STUDY: WAFER LOGISTICS

To demonstrate the applicability of the modular specification approach we use the model of a lithography system from ASML [15]. Lithography systems are manufacturing systems which process integrated circuits (ICs) by exposing reticle images on silicon wafers. Figure 5 depicts the wafer flow, the 5 operational units (the Conditioner (COND), the Discharge Unit (DU), the Pre-Aligner (PA) and two chucks (CH0 and CH01)) and 2 transport units (the Unload and Load Robots (UR and LR)). The arrows in the figure model the possible flows wafer can have in the system. We do not elaborate on the modeling of the resources, peripherals and actions of the example lithography machine and refer to [15] for the details thereof. Instead our starting point is the set of Activities provided by the system. This list is given in Table 1. To avoid clutter in the figures and text we will refer to these activities using bold-case letters as indicated in Table 1 (e.g. activity $Track\_2\_COND$ is refered to be letter **a**). We denote the corresponding activity on a specific wafer with an index i starting from 1 and ending with n, where n is the size of the batch (e.g. **a**_i refers to activity $Track\_2\_COND$ performed on behalf of wafer i).

For every product we define its life-cycle, i.e. the operation flow. Figure 4 depicts the logistics automata **Life-Cycle** which captures the operation flow of a product. The life-cycle captures the input (**a**), pre-processing (**b,c,d,e** and **f**), measure/exposure (**g,h,i,j** or **k,l,m,n**) and output (**o**) of a wafer in the system. The pre-processing activities are necessary to set a ideal wafer temperature for an accurate exposure process and to align the wafer for a correct overlay of multiple exposures. For performance gains the expose stage is composed of two identical wafer chucks (CH0/CH1) with swap

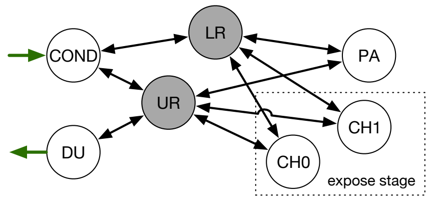

Figure 5: Routing and resources of the Wafer Logistics.

Table 1: Set of activities for our case study.

a:Track_2_COND	**e**:PA_PreAlign	**k**:LR_2_CH1
b:COND_Conditioning	**g**:LR_2_CH0	**l**:CH1_Measure
c:COND_2_UR	**h**:CH0_Measure	**m**:CH1_Expose
d:UR_2_PA	**i**:CH0_Expose	**n**:CH1_2_UR
f:PA_2_LR	**j**:CH0_2_UR	**o**:UR_2_DU
p(q):CH0(1)_M_2_E	**r(s)**:CH0(1)_E_2_M	

Table 2: Size of the logistics automaton and optimization-space of a batch of 25 wafers.

	Logistics Automaton		Optimization-Space		
Model	N. States	N. Trans.	N. States	N. Trans.	Makespan (s)
Logistics	DNF	DNF	DNF	DNF	-
+Capacity	DNF	DNF	DNF	DNF	-
+FIFO	64876	206929	DNF	DNF	-
+Swap	10566	30421	128412	365711	337
+Exchange	9060	26099	58287	166983	337

activities (**p,q,r** and **s**) to alternate between the measure/expose activities. Thus this measure/exposure operation is captured by two possible sequences of activities (**g,h,p,i,r,j** or **k,l,q,m,s,n**) each modeling the measure/exposure on a specific chuck (visible from the branching in automaton **Life-Cycle**). For a batch of n products we specify n instances of such automata and composed them using the $\odot$ operator. All wafers follow the same life-cycle in this case study (but our approach is general enough to deal with different operational flows for different products).

Next to the **Life-Cycle** requirement we specify other system requirements. These requirements are captured by the **Capacity**, **Swap**, **First-In** (FI) and **First-Out** (FO) constraint depicted in Figure 4. Note that all these requirements are specified as deterministic constraint automata. **Capacity** constraints enforces that each resource has limited capacity (in this case all resource have unary capacity). The **Swap** constraint ensures that between two consecutive measure or expose activities the chucks first have to swap. The **First-In** and **First-Out** requirements enforce the order of processing of the wafers in a batch. These automata define the *system specification* as required for a correct manufacturing.

Table 2 shows the results in terms of the number of states (N. States) and transitions (N. Trans.) of the resulting logistics and (max,+) automaton state-spaces (Logistics Automaton and Optimization-space) as well as the obtained minimal makespan (Makespan). These results are computed using 8 Intel i7 920@2.67Ghz CPUs with 8GB of memory. The models and the composition and constraining operators are implemented using the CIF (Compositional Interchange Format) tooling [13]. The values shown in Table 2 correspond to a batch of 25 wafers. DNF (Did Not Finish) denotes the case where the construction of the logistics automaton (or its optimization-space) failed to finish. The first row (Logistics) represents the unconstrained system (only the **Life-cycle** is specified). Every row adds an additional constraining on top of the previous ones (indicated by the + symbol). Note that the incremental constraining of the system with requirements **Capacity**, **First-In** and **First-Out** (FIFO) and **Swap** systematically reduces the size of the worst-case optimization-space (according to the results of Section 6). Note that in this case the reduction occurs even on the state-spaces of the logistics automata and on the actual optimization-space until the point that the optimization-space can be built and explored to find the minimal makespan of 337 seconds. We leave the study to identify the conditions for which the actual optimization-space reduces as part of future work.

To display the use of the approach as a formal means to consciously use over-specification to further prune the optimization-space we use domain knowledge to generate one additional non-essential constraint called **Exchange** (see Figure 4). The **Exchange** constraint enforces an activity order between the load of wafer (i+2) to a chuck and unload of wafer (i) from the same chuck. The results are shown in Table 2. Notice that in this case an the optimal solution could already be found without over-specification, Because of this we can compare quantitatively the impact of the additional constraint: it effectively reduces the optimization-space by more than 50%. In general a deterministic constraints results in an upperbound of the optimal makespan (see Section 6); in this case the optimal result is actually preserved.

8 RELATED WORK

To the best of our knowledge, this is the first paper in which modular formal requirements are proposed as an effective means to get qualitative grip on state-spaces sizes.

This work combines the ingredients from the modular constraint-oriented approach of the LOTOS framework [3, 7] and of Supervisory Control Theory (SCT) [10] developed in [14] and applied in [4, 5, 12]. We build upon the general concepts of SCT, but refine this framework by allowing only non-recursive requirements and by explicitly distinguishing logistics automata from constraint automata.

To get insight in the impact of automata composition on the state-space sizes we took inspiration from the Calculus of Communication Systems (CCS) [8], in particular from Milner's simulation relation $\prec$. This relation captures behavior but abstracts from all structural information of the automata. We strengthen this relation by adding structural information (in the form of an injectivity requirement) to allow qualitative reasoning about state-space sizes. In addition it benefits from the effective proof technique of establishing simulation relations. The strengthening of pre-order $\prec$ makes $\sqsubseteq$ into a partial-order. As far as we have been able to verify, this is the first paper in which the inclusion relation $\sqsubseteq$ (to compare transitions systems in a structural way) is established.

9 CONCLUSIONS

In this paper we introduced a framework for the specification and optimization of batch-oriented manufacturing systems and show that their makespan optimization is NP-Hard. We introduce a composition operator for logistics automata to address the specification of complex manufacturing system in a modular fashion. We further introduce the notion of constraint automata to capture requirements across different products flows such as input orderings, capacity and safety constraint. Using a constraint operator we are able to modularly compose deterministic constraints with logistics automata and show that this operation efficiently prunes the (worst-case) optimization-space. Moreover, we show that our approach gives explicit control over the worst-case size of the optimization-space. We provide proof of these results by defining an inclusion relation between logistics automata which relates qualitatively their state-space sizes. Moreover, obtained solutions are i) makespan optimal with respect to the added constraints and ii) upperbounds with respect to the original optimization problem. As future work, we are interested in establishing the conditions for which the constraining operation prunes the *actual* optimization-space.

ACKNOWLEDGMENTS

This research is supported by the Dutch Technology Foundation NWO-AES (Applied and Engineering Sciences), carried out as part of the Robust Cyber-Physical Systems (RCPS) program, project number 12694.

REFERENCES

[1] Francois Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. 2001. Synchronization and linearity: an algebra for discrete event systems.

[2] J. Bastos, B. van der Sanden, O. Donk, J. Voeten, S. Stuijk, R. Schiffelers, and H. Corporaal. 2017. Identifying bottlenecks in manufacturing systems using stochastic criticality analysis. In *2017 Forum on Specification and Design Languages (FDL)*. 1–8. https://doi.org/10.1109/FDL.2017.8303901

[3] Ed Brinksma. 1990. Constraint-oriented specification in a constructive formal description technique. In *Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness*, J. W. de Bakker, W. P. de Roever, and G. Rozenberg (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 130–152.

[4] Esmée L. G. Bertens et al. 2009. Supervisory control synthesis for exception handling in printers.

[5] Stefan T. J. Forschelen, Joanna M. van de Mortel-Fronczak, Rong Su, and Jacobus E. Rooda. 2012. Application of supervisory control theory to theme park vehicles. *Discrete Event Dynamic Systems* 22, 4 (01 Dec 2012), 511–540. https://doi.org/10.1007/s10626-012-0130-6

[6] Richard M. Karp. 1972. *Reducibility among Combinatorial Problems*. Springer US, Boston, MA, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[7] Guy Leduc. 1992. A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems* 25, 1 (1992), 23 – 41. https://doi.org/10.1016/0169-7552(92)90122-7 Formal Description Techinique (FDT) Concepts and Tools.

[8] Robin Milner. 1983. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science* (1983).

[9] Joost Van Pinxten, Umar Waqas, Marc Geilen, Twan Basten, and Lou Somers. 2017. Online Scheduling of 2-Re-entrant Flexible Manufacturing Systems. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 160 (Sept. 2017), 20 pages. https://doi.org/10.1145/3126551

[10] P. J. Ramadge and W. M. Wonham. 1984. Supervisory control of a class of discrete event processes. In *Analysis and Optimization of Systems*, A. Bensoussan and J. L. Lions (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 475–498.

[11] Ofira Shmueli, Lior Fink, and Nava Pliskin. 2012. Over-Requirement in Software Development: an Empirical Investigation of the 'IKEA' Effect.. In *ECIS*. 85.

[12] R. J. M. Theunissen et al. 2014. Application of Supervisory Control Synthesis to a Patient Support Table of a Magnetic Resonance Imaging Scanner. *IEEE Transactions on Automation Science and Engineering* (2014).

[13] Dirk A. van Beek, Wan Fokkink, D. Hendriks, A. Hofkamp, Jasen Markovski, J. M. van de Mortel-Fronczak, and Michel A. Reniers. 2014. CIF 3: Model-Based Engineering of Supervisory Controllers. In *TACAS*.

[14] B. van der Sanden, J. Bastos, J. Voeten, M. Geilen, M. Reniers, T. Basten, J. Jacobs, and R. Schiffelers. 2016. Compositional specification of functionality and timing of manufacturing systems. In *2016 Forum on Specification and Design Languages (FDL)*. 1–8. https://doi.org/10.1109/FDL.2016.7880372

[15] B. van der Sanden, M. Reniers, M. Geilen, T. Basten, J. Jacobs, J. Voeten, and R. Schiffelers. 2015. Modular model-based supervisory controller design for wafer logistics in lithography machines. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 416–425. https://doi.org/10.1109/MODELS.2015.7338273