Taming the state-space explosion in the makespan optimization of Flexible Manufacturing Systems

JOÃO BASTOS, Eindhoven University of Technology, The Netherlands
JEROEN VOETEN, Eindhoven University of Technology, The Netherlands
SANDER STUIJK, Eindhoven University of Technology, The Netherlands
RAMON SCHIFFELERS, Eindhoven University of Technology and ASML, The Netherlands
HENK CORPORAAL, Eindhoven University of Technology, The Netherlands

This article presents a modular automaton-based framework to specify flexible manufacturing systems and to optimize the makespan of product batches. The Batch Makespan Optimization (BMO) problem is NP-Hard and optimization can therefore take prohibitively long, depending on the size of the state-space induced by the specification. To tame the state-space explosion problem we develop an algebra based on automata equivalence and inclusion relations that consider both behaviour and structure. The algebra allows us to systematically relate the languages induced by the automata, their state-space sizes and their solutions to the BMO problem. Further we introduce a novel constraint-based approach to systematically prune the state-space based on the the notions of nonpermutation-repulsiveness and permutation-attractiveness. We prove that constraining a nonpermutation-repulsing automaton with a permutation-attracting constraint always reduces the state-space. This approach allows us to i) compute optimal solutions of the BMO problem when the (additional) constraints are taken into account and ii) compute bounds for the (original) BMO problem (without using the constraints). We demonstrate the effectiveness of our approach by optimizing an industrial wafer handling controller.

CCS Concepts: • **Theory of computation** → **Formal languages and automata theory**; • **Computing methodologies** → **Model verification and validation**; • **Computer systems organization** → *Embedded and cyber-physical systems*;

ACM Reference Format:
João Bastos, Jeroen Voeten, Sander Stuijk, Ramon Schiffelers, and Henk Corporaal. 2020. Taming the state-space explosion in the makespan optimization of Flexible Manufacturing Systems. *ACM Transactions on Cyber-Physical Systems* 1, 1, Article 1 (January 2020), 25 pages. https://doi.org/10.1145/3426194

1 INTRODUCTION

Flexible Manufacturing Systems (FMSs) are Cyber-Physical Systems which perform operations on batches of products. Production and transportation resources (the physical part) carry out operations while a logistics controller (the cyber part) assigns operations to resources and determines their order. Examples of these systems include lithography machines and industrial production printers. During their design, functional requirements (such as keeping the order of products in a batch or

This research is supported by the Dutch NWO-TTW, carried out as part of the Robust Cyber-Physical Systems (RCPS) program, project number 12694. Research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement no 826452 (project Arrowhead Tools).
The proofs for all Lemmas and Theorems can be found in an Appendix provided with the supplementary material.

XXXX-XXXX/2020/1-ART1 $15.00
https://doi.org/10.1145/3426194

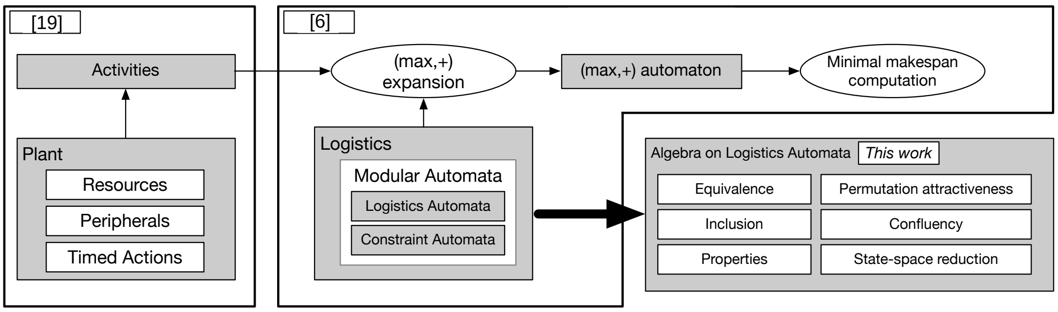


Fig. 1. Overview of ingredients from previous work that this paper uses and extends upon.

avoiding collisions) must be satisfied while at the same time demanding performance requirements must be met, for instance regarding productivity.

In [6, 19] a framework is proposed for the batch-oriented specification and design exploration of FMSs (shown in Figure 1). The novelty of this framework lies in the separation of concerns between specification and optimization, together with its compositional support for requirements specification. The physical system is abstracted into sets of resources, peripherals and actions. Different pieces of end-to-end deterministic functional behavior are captured as *activities* which are Directed Acyclic Graphs (DAGs) composed of multiple peripheral actions and dependencies among them. The temporal behavior of each activity is characterized using (max,+) algebra. An *activity sequence* can be constructed to describe more elaborate functional behaviors by considering multiple activities, such as the complete manufacturing of a product where each operation is captured by a single activity. The set of all possible activity sequences is captured as the language accepted by a finite state automaton, called a *logistics automaton.* Logistics automata can be defined modularly for each product in a batch. The logistics automaton of the batch itself is then obtained by composing these automata by means of a *composition operator.* Besides logistics requirements a system also exhibits constraints on different product flows, such as resource capacity and safety constraints. These are modularly captured in our framework as *constraint automata* and a *constraint* operator is provided to compose them with logistics automata.

Once the FMS has been specified, the minimal completion time of a batch of products can be computed by solving an optimization problem called the Batch Makespan Optimization (BMO) problem. This BMO problem is NP-Hard [6]. Computing makespan-optimal solutions can therefore take prohibitively long, due to state-space explosion. To keep state-spaces manageable, we complement the work of [6] in this article by developing an algebra of logistics automata based on *equivalence* and *inclusion* relations that take both behavioural and structural information into account. The algebra allows us to relate the languages induced by the automata, their state-space sizes and their solutions to the BMO problem, thereby providing a systematic approach to specify and explore different designs of FMSs. To the best of our knowledge, this is the first work in which an algebraic framework is developed to reason qualitatively about state-space sizes of FSMs. Further we introduce a novel constraint-based approach to systematically prune the state-space based on the notions of *nonpermutation-repulsiveness* and *permutation-attractiveness.* We prove that constraining a nonpermutation-repulsing automaton with a permutation-attracting constraint always reduces the state-space. This approach allows us to i) compute optimal solutions of the BMO problem when the (additional) constraints are taken into account and ii) compute bounds for the (original) BMO problem (without using the constraints). The approach is inspired by industrial practices, where manufacturing systems are typically over-specified [17] and in which over-specification is used

implicitly and unconsciously to deal with complexity. Examples of over-specification encountered in industrial cases are disallowing multiple mapping possibilities for an operation or enforcing the static ordering of system operations. This approach enables system designers to consciously exploit over-specification by an explicit introduction of constraints to reduce the complexity of computing a solution to the BMO problem. We will demonstrate the effectiveness of our framework by specifying and optimizating an industrial wafer handling controller.
Overview: The remainder of this article is organized as follows. Section 2 and 3 summarize the framework that we will extend in this article. In Section 4 we define equivalence and inclusion relations on logistics automata that take into account both behavioural and structural information, and we establish their key algebraic properties. Section 5 establishes sufficient conditions to ensure that automata constraining reduces state-space sizes. Section 6 demonstrates the use of the algebra to systematically relate automata languages, their state-space sizes, and their solutions to the BMO problem. In Section 7 the framework is applied to specify and optimize an industrial wafer handling controller. Related work is discussed in Section 8 and Section 9 concludes the article.

2 SPECIFICATION FRAMEWORK

This article extends the specification framework of [19] (highlighted in Figure 1). In this section, we summarize the ingredients of this framework and its essential properties.

2.1 Plant and Activities

A manufacturing system is decomposed into a plant as 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 2 a) depicts a synthetic example of a system decomposed into resources R1, R2 and R3, peripherals p1, p2, p3, p4 and p5 and actions x_1, x_2, x_3, x_4, x_5, x_6 ands x_7. Using 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 2 b) depicts two such activities, a and b. By sequencing multiple activities to form an *activity sequence* we capture more elaborate operational behavior, such as the complete manufacturing of a product. We denote an

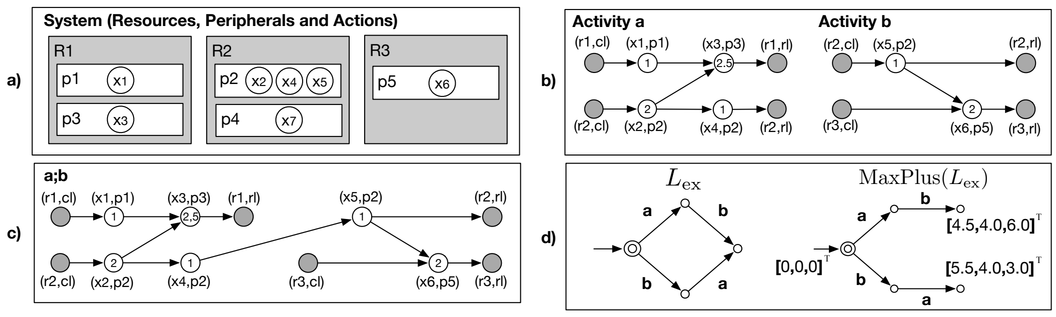


Fig. 2. Overview of the concepts of the framework of [19]: a) an FMS decomposed into resources R1, R2 and R3, peripherals p1, p2, p3, p4 and p5 and actions x_1, x_2, x_3, x_4, x_5, x_6 ands x_7, b) two example activities a and b, c) sequenced activity $a;b$ and d) example logistics automaton L_{ex} and (max,+) automaton MaxPlus(L_{ex}).

activity sequence as $\underline{a} = a_1\ a_2\ \cdots\ a_n$ where $a_1, a_2, \cdots, a_n$ denote activities. The temporal behavior of an activity sequence can be determined by the *sequencing operator* (;) [19] that combines two activities in 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 2 c) depicts the result of sequencing activities a and b, 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. To this end, a function $T : \mathcal{A} \longrightarrow \mathbb{R}_{\geq 0}$ maps each action to its (fixed) execution time. 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 2 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 $\gamma_R : \mathcal{R} \longrightarrow \mathbb{R}^{-\infty}$ 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 (here *availability* does not represent resource capacity, but timing availability) time of resource r. For example, in Figure 2 d), the initial system state is $\gamma_R = [0, 0, 0]^T$, implying that all system resources are available at time 0.

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, taking into account the dependencies and execution times of its nodes. In the example of Figure 2 d), the resource-time stamp vector after executing activities a and b is $\gamma'_R = [4.5, 4.0, 6.0]^T$, thus for the next activity resources $R1$, $R2$ and $R3$ are available at times 4.5, 4.0 and 6.0, respectively.

It is well-know that (max,+) algebra can be used to mathematically describe timed synchronous systems [5, 9, 13]. This observation also applies to activities and in [19] 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. In case a sequence $a_1\ a_2\ \cdots\ a_n$ of activities is executed, we thus have $\gamma'_R = M_{a_1;a_2;\cdots;a_n} \otimes \gamma_R$ (where ; denotes the activity sequencing operator). It is not difficult to see that this expression is equivalent to $M_{a_n} \otimes \cdots \otimes M_{a_2} \otimes M_{a_1} \otimes \gamma_R$. For a good introduction to (max,+) algebra we refer the reader to [1, 5, 15]. To determine the makespan of an activity sequence, we assume the system to start with resource availability vector $\mathbf{0}_R$, i.e. for which $\mathbf{0}_R(r) = 0$ for each $r \in \mathcal{R}$. The makespan is determined by the $(max, +)$ norm of the state after execution. The makespan of activity sequence $\underline{a} = a_1\ a_2\ \cdots\ a_n$, written $mks(\underline{a})$, is therefore given by $\|M_{a_n} \otimes \cdots \otimes M_{a_2} \otimes M_{a_1} \otimes \mathbf{0}_R\|$ (where $\|\cdot\|$ denotes the norm operator). As an example we consider the sequenced activity $a;b$ of Figure 2 c) again. The makespan of the corresponding activity sequence $a\ b$ is given by:

$$mks(a\ b) = M_b \otimes M_a \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & -\infty & -\infty \\ -\infty & 1 & -\infty \\ -\infty & 3 & 2 \end{bmatrix} \otimes \begin{bmatrix} 3.5 & 4.5 & -\infty \\ -\infty & 3 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 4.5 \\ 4 \\ 6 \end{bmatrix}.$$

As a prelude to Section 2.2, Figure 2 d) presents a logistics automaton L_{ex} encoding two activity sequences $a\ b$ and $b\ a$. To determine the activity sequences with the smallest makespan, the (max,+) automaton MaxPlus(L_{ex}) of L_{ex} is computed by decorating the states of L_{ex} with appropriate resource availability vectors. Activity sequences that terminate in states with the smallest vector norms are makespan-optimal, which is proven in Section 3. For the automaton L_{ex} in Figure 2 d) this happens to be sequence $b\ a$.

2.2 Logistics

An activity sequence can model the *complete* manufacturing of a product or batch of products, where a single activity models one manufacturing operation. In general, more than one activity sequence will satisfy the requirements imposed on the system. The set of all activity sequences that satisfy these requirements is encoded by a *logistics automaton.*

Definition 1 (Logistics automaton). *A logistics automaton is a tuple $\langle \mathcal{S}, \mathcal{A}ct, \dashrightarrow, \mathcal{S}_0 \rangle$, where $\mathcal{S}$ is a finite (possibly empty) set of states, $\mathcal{A}ct$ is a finite (possibly empty) set of activities, $\dashrightarrow \subseteq \mathcal{S} \times \mathcal{A}ct \times \mathcal{S}$ is a transition relation, and $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states. Here $\mathcal{S}_0 = \{s_0\}$ except when $\mathcal{S} = \emptyset$. In that case the automaton is empty and therefore $\mathcal{S}_0 = \emptyset$ as well. Let $s \xrightarrow{a} s'$ be a shorthand for $(s, a, s') \in \dashrightarrow$. The following additional properties must hold:*

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

A logistics automaton encodes a collection of activity sequences. This collection is called the language of the automaton.

Definition 2 (Language of a logistics automaton). *Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dashrightarrow, \mathcal{S}_0 \rangle$ be a logistics automaton. The language $\mathcal{L}(L)$ of L is defined by*

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

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 $s_1, ..., s_n \in \mathcal{S}$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n = s'$. Further $s \xrightarrow{\underline{a}}$ denotes that $s \xrightarrow{\underline{a}} s'$ for some $s' \in \mathcal{S}$, $s \dashrightarrow$ denotes that $s \xrightarrow{a} s'$ for some $a \in \mathcal{A}ct$ and $s' \in \mathcal{S}$ and $s \not\dashrightarrow$ denotes that $s \dashrightarrow$ 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". This means it should finish in a final *state, i.e. a state with no outgoing transitions. As a consequence languages of logistics automata are not prefix closed in general.*

Example 1. *Recall Figure 2 d) depicting logistic automaton L_{ex}. Nodes represent states and edges represent transitions. Activities are annotated on edges and the initial state is distinguished by an extra circumference. L_{ex} encodes the language $\mathcal{L}(L_{ex}) = \{a\ b, b\ a\}$.*

2.3 Modular Logistics Specification: composition and constraining

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. For this purpose a composition operator on logistics automata is provided. The operator allows the manufacturing of the products to be specified individually and then composed to obtain an automata that encodes all the logistics requirements for the full batch of products.

Definition 3 (Composition of Logistics Automata). *Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dashrightarrow_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dashrightarrow_2, S_{0_2} \rangle$ be logistics automata. Before we define the composition automaton $L_1 \odot L_2$, we first define relation $\dashrightarrow \subseteq (\mathcal{S}_1 \times \mathcal{S}_2) \times (\mathcal{A}ct_1 \cup \mathcal{A}ct_2) \times (\mathcal{S}_1 \times \mathcal{S}_2)$ as the smallest set V satisfying the following inference rules:*

$$\frac{s \xrightarrow{a}_1 s' \quad a \in \mathcal{A}ct_1 \backslash \mathcal{A}ct_2}{(s, t) \xrightarrow{a} (s', t)} \ (1) \qquad \frac{s \xrightarrow{a}_1 s' \quad t \xrightarrow{a}_2 t' \quad a \in \mathcal{A}ct_1 \cap \mathcal{A}ct_2}{(s, t) \xrightarrow{a} (s', t')} \ (2)$$

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

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

$$\mathcal{S} = \begin{cases} \emptyset, & \text{if } \mathcal{S}_1 = \emptyset \text{ or } \mathcal{S}_2 = \emptyset \\ \{(s,t) \in \mathcal{S}_1 \times \mathcal{S}_2 \mid (s_{0_1}, s_{0_2}) \dashrightarrow^* (s,t) \\ \text{and for some } (s',t') \in \mathcal{S}_1 \times \mathcal{S}_2 \text{ with} \\ s' \not\dashrightarrow_1 \text{ and } t' \not\dashrightarrow_2, (s,t) \dashrightarrow^* (s',t')\} & \text{if } \mathcal{S}_{0_1} = \{s_{0_1}\} \text{ and } \mathcal{S}_{0_2} = \{s_{0_2}\} \end{cases}$$

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

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

The composition automaton $L_1 \odot L_2$ is now defined as $\langle \mathcal{S}, \mathcal{A}ct_1 \cup \mathcal{A}ct_2, \dashrightarrow', \mathcal{S}_0 \rangle$.

The language of a composite logistics automaton can be computed from the languages of its constituent automata by merging together the sequences of the languages of these constituents:

Lemma 1. *Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dashrightarrow_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dashrightarrow_2, S_{0_2} \rangle$ be logistics automata. Then $\mathcal{L}(L_1 \odot L_2) = \{\underline{a} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^* \mid \underline{a} \backslash \mathcal{A}ct_1 \in \mathcal{L}(L_1) \text{ and } \underline{a} \backslash \mathcal{A}ct_2 \in \mathcal{L}(L_2)\}$ where $\underline{a} \backslash \mathcal{A}ct_1$ and $\underline{a} \backslash \mathcal{A}ct_2$ denote the projections of activity sequence $\underline{a}$ onto alphabets $\mathcal{A}ct_1$ and $\mathcal{A}ct_2$ respectively.*

By means of the composition operator we can specify the behavior of a batch of products in a modular way. This operator respects the requirements specified for the individual products and ensures the completion of each of them. For a complete specification of the manufacturing of a batch of products, we also need to specify constraints across different product flows 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. Such constraints are expressed in terms of *constraint automata.*

Definition 4 (Constraint automaton). *A constraint automaton is a tuple $\langle \mathcal{S}, \mathcal{A}ct, \dashrightarrow, \mathcal{S}_0 \rangle$, where $\mathcal{S}$ is a finite (possibly empty) set of states, $\mathcal{A}ct$ is a finite (possibly empty) set of activities, $\dashrightarrow \subseteq \mathcal{S} \times \mathcal{A}ct \times \mathcal{S}$ 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:*

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

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

Definition 5 (Language of a constraint automaton). *Let $C = \langle \mathcal{S}, \mathcal{A}ct, \dashrightarrow, \mathcal{S}_0 \rangle$ be a constraint automaton. The language $\mathcal{L}(C)$ of C is defined by*

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

Notice that constraint automata are distinct from logistics automata in the sense that they can be recursive and are therefore able to encode infinite languages. Constraints can be applied to logistics automata through the constraint operator. A constraint automaton $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dashrightarrow_2, \mathcal{S}_{0_2} \rangle$ is called *a constraint on* logistics automaton $L = \langle \mathcal{S}_1, \mathcal{A}ct_1 \dashrightarrow_1, \mathcal{S}_{0_1} \rangle$ if $\mathcal{A}ct_2 \subseteq \mathcal{A}ct_1$. Applying constraint C to automaton L yields a new logistics automaton which is denoted by $L \upharpoonright C$.

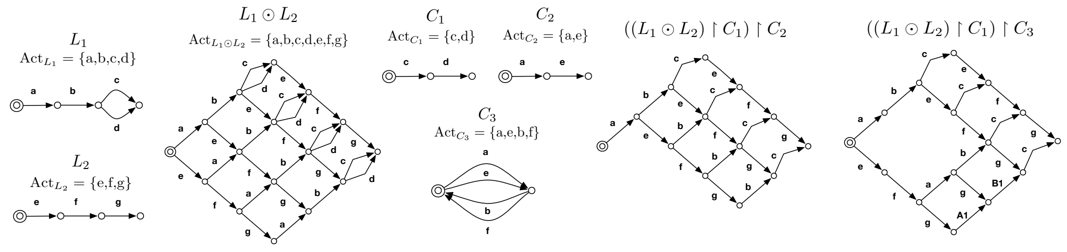

L_1
$\mathrm{Act}_{L_1} = \{a,b,c,d\}$
L_2
$\mathrm{Act}_{L_2} = \{e,f,g\}$
$L_1 \odot L_2$
$\mathrm{Act}_{L_1 \odot L_2} = \{a,b,c,d,e,f,g\}$
C_1
$\mathrm{Act}_{C_1} = \{c,d\}$
C_2
$\mathrm{Act}_{C_2} = \{a,e\}$
C_3
$\mathrm{Act}_{C_3} = \{a,e,b,f\}$
$((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_2$
$((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_3$
B1
A1

Fig. 3. Composing logistics automata L_1 and L_2 and constraining them with constraints C_1, C_2 and C_3.

Definition 6 (Constraint Operator). *Let $L = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ be a logistics automaton and $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ 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 $\dot{\rightarrow} \subseteq (\mathcal{S}_1 \times \mathcal{S}_2) \times \mathcal{A}ct_l \times (\mathcal{S}_1 \times \mathcal{S}_2)$ as the smallest set V satisfying the following inference rules:*

$$\frac{s \xrightarrow{a}_1 s' \quad a \in \mathcal{A}ct_1 \backslash \mathcal{A}ct_2}{(s,t) \xrightarrow{a} (s',t)} \ (1) \qquad \frac{s \xrightarrow{a}_1 s' \quad t \xrightarrow{a}_2 t' \quad a \in \mathcal{A}ct_1 \cap \mathcal{A}ct_2}{(s,t) \xrightarrow{a} (s',t')} \ (2)$$

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

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

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

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

The constrained automaton $L \upharpoonright C$ is now defined as $\langle \mathcal{S}, \mathcal{A}ct_1, \dot{\rightarrow}', \mathcal{S}_0 \rangle$.

Both the constraint and composition operators assume multi-way synchronization for transitions. Note however that the constraint operator requires the logistics automaton to run to completion (i.e. reaches a final state), 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 a batch). On the contrary, the composition operator requires both logistics automata to run to completion. The language of a constrained logistics automaton can be computed from the languages of its constituent automata by filtering out activity sequences that are not consistent with the constraint:

Lemma 2. *Let $L = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ be a logistics automaton and let $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ be a constraint on L. Then $\mathcal{L}(L \upharpoonright C) = \{\underline{a} \in \mathcal{L}(L) \mid \underline{a} \backslash \mathcal{A}ct_2 \in \mathcal{L}(C)\}$.*

Finally we claim that the application of a constraint to a logistics automaton, results in a subset of the original language:

Lemma 3 (Language constraining). *Let L be a logistics automaton and let C be a constraint on L. Then $\mathcal{L}(L \upharpoonright C) \subseteq \mathcal{L}(L)$.*

Example 2. *Consider the automata depicted in Figure 3. L_1 and L_2 represent two distinct product flows and $L_1 \odot L_2$ represents their combined execution flow in a two-product batch. Constraint C_1 represents a requirement on the order of activities c and d on the product modeled by L_1. Constraints C_2 and C_3 express requirements concerning the product flows of $L_1 \odot L_2$. $((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_2$ shows the application of constraints C_1 and C_2 to $L_1 \odot L_2$. Since C_1 requires activity d to be preceded by c, transition d is removed. Further all behavior in which activity e is not preceded by a is removed due to C_2. $((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_3$ shows the application of constraint C_1 and C_3 to $L_1 \odot L_2$. In this case all behavior in which activities b or f and a or e do not occur in alternating order is removed (for instance sequence a e b c f g is removed). Notice that, consistent with Lemma 3, $\mathcal{L}(((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_2) \subseteq \mathcal{L}(L_1 \odot L_2)$ and $\mathcal{L}(((L_1 \odot L_2) \upharpoonright C_1) \upharpoonright C_3) \subseteq \mathcal{L}(L_1 \odot L_2)$.*

3 BATCH MAKESPAN OPTIMIZATION

We are interested in finding the activity sequence with the lowest makespan in the language of a logistics automaton. In [6] we formalized this Batch Makespan Optimization (BMO) problem and proved it to be NP-hard:

Definition 7 (Batch Makespan Optimization (BMO)). *Given a Logistics automaton L determine an $\underline{a} \in \mathcal{L}(L)$ such that $mks(\underline{a}) \leq mks(\underline{a}')$, for all $\underline{a}' \in \mathcal{L}(L)$.*

Lemma 4. *The Batch Makespan Optimization problem is NP-hard.*

Through language inclusion we can establish sub-optimal solutions to the BMO problem and find bounds on the optimal makespan:

Lemma 5. *Let L_1 and L_2 be logistics automata for which $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ and assume $\underline{a}$ to be a BMO solution to L_1 and $\underline{a}'$ to be a BMO solution to L_2. Then $mks(\underline{a}) \geq mks(\underline{a}')$.*

3.1 Solving the BMO problem: (max,+) automata

The BMO 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. The result of this expansion is a new logistics automaton where each state includes a resource time-stamp vector capturing the resource availability after executing the activities in the path leading to that state. Given a logistics automaton L we call its timed expansion a *(max,+) automaton* and denote it by MaxPlus(L).

Definition 8 ((max,+) automaton). *Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dashrightarrow, \mathcal{S}_0 \rangle$ be a logistics automaton. First define MaxPlusStates(L) as the smallest set V satisfying inference rules (1) and (2):*

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

Here γ_R denotes a resource time-stamp vector and $\mathbf{0}_R$ denotes the resource time-stamp vector containing only 0 valued entries. Note that $\mathbf{0}_R$ represents the initial system state in which all resources are available at time zero. $\mathbf{M}_a$ denotes the $(max, +)$ matrix corresponding to activity $a \in \mathcal{A}ct$ and $s, s' \in \mathcal{S}$. Then we define MaxPlus(L) as $(MaxPlusStates(L), \mathcal{A}ct, \dashrightarrow', \mathcal{S}_0')$, where $\rightarrow' = \left\{(s, \gamma_R), a, (s', \gamma_R') \in MaxPlusStates(L) \times \mathcal{A}ct \times MaxPlusStates(L) \mid s \xrightarrow{a} s' \text{ and } \gamma_R' = M_a \otimes \gamma_R\right\}$ and $\mathcal{S}_0' = \emptyset$ if $\mathcal{S}_0 = \emptyset$ and $\mathcal{S}_0' = \{(s_0, \mathbf{0}_R)\}$ otherwise.

Each state of the logistics automaton can occur multiple times in the (max,+) automaton, depending on the cumulative products of (max,+) activity matrices along the paths from the initial state leading to this particular state. Therefore the number of states of the (max,+) automaton is at least as large as the number of states of the corresponding logistics automaton. Their languages are however equivalent, which is stated by the following Lemma.

Lemma 6. *Given a logistics automaton L then $\mathcal{L}(L) = \mathcal{L}(MaxPlus(L))$.*

The state-space encoded by a (max,+) automaton includes all the necessary functional and temporal information of the system in terms of allowed activity sequences as well as their respective completion times. For this reason, we denote the state-space of a (max,+) automaton the *optimization-space*. The second step to solve the BMO problem is to explore all the final states (states with no outgoing transitions) of the optimization-space and comparing the norms of the corresponding resource-time stamp vectors. Any sequence in the optimization-space leading to a final state with the lowest occurring norm is a solution to the BMO problem as stated in the following theorem.

Lemma 7. *Let L be a logistics automaton and $MaxPlus(L) = \{\mathcal{S}, \dot{\rightarrow}, \mathcal{A}ct, \mathcal{S}_0\}$ its corresponding (max,+) automaton. Let $(s, \gamma) \in \mathcal{S}$ be such that $(s, \gamma) \not\rightarrow$ and that for all $(s', \gamma') \in \mathcal{S}$ with $(s', \gamma') \not\rightarrow$, $\|\gamma\| \leq \|\gamma'\|$, and let $\underline{a} \in \mathcal{L}(L)$ be such that $(s_0, \mathbf{0}_R) \xrightarrow{\underline{a}} (s, \gamma)$. Then for all $\underline{a}' \in \mathcal{L}(L)$: $mks(\underline{a}) \leq mks(\underline{a}')$.*

Example 3. *Consider the (max,+) automaton of logistics automaton L_{ex} depicted in Figure 2. Consistent with Lemma 6, indeed $\mathcal{L}(MaxPlus(L_{ex})) = \mathcal{L}(L_{ex})$. The temporal execution of activity sequences $a\ b$ and $b\ a$ lead to different resource time-stamp vectors and thus the final state of L_{ex} is duplicated in $MaxPlus(L_{ex})$. According to Lemma 7, activity sequences in $\mathcal{L}(L_{ex})$ are makespan-optimal if they terminate in the states with the smallest vector norms in $MaxPlus(L_{ex})$. The final state in $MaxPlus(L_{ex})$ with the smallest vector norm is labelled with vector $[5.5, 4.0, 3.0]^T$. This final state corresponds to sequence $b\ a$. Hence $b\ a$ is a solution to the BMO problem of L_{ex}.*

3.2 Worst-case optimization-space: Tree automata

The size of the state-space of a (max,+) automaton is at least as large as that of the corresponding logistics automaton, a fact that we will prove in the next section. It can even grow exponentially in the size of the state-space of the corresponding logistics automaton, which is the very reason that the BMO problem is NP-hard. In the worst case each path in the logistics automaton induces a unique cumulative product of (max,+) matrices. This worst case is captured by what we will call the *tree automaton* Tree(L) of L, the proof of which is given in the next section.

Definition 9 (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 $(Paths(L), \mathcal{A}ct, \dot{\rightarrow}', \mathcal{S}_0')$, where $\dot{\rightarrow}' = \{(q\ s, a, q\ s\ a\ s') \in Paths(L) \times \mathcal{A}ct \times Paths(L) \mid s \xrightarrow{a} s'\}$ and $\mathcal{S}_0' = \emptyset$ if $\mathcal{S}_0 = \emptyset$ and $\mathcal{S}_0' = \{s_0\}$ otherwise.

Just like the MaxPlus operator, the Tree operator leaves the language of the logistics automaton unaltered.

Lemma 8. *Given a logistics automaton L then $\mathcal{L}(L) = \mathcal{L}(Tree(L))$.*

4 ALGEBRA ON LOGISTICS AUTOMATA

So far we introduced the $\odot$, $\upharpoonright$, MaxPlus and Tree operators on logistics automata. In this section we complement these operators with equivalence and inclusion relations and prove important

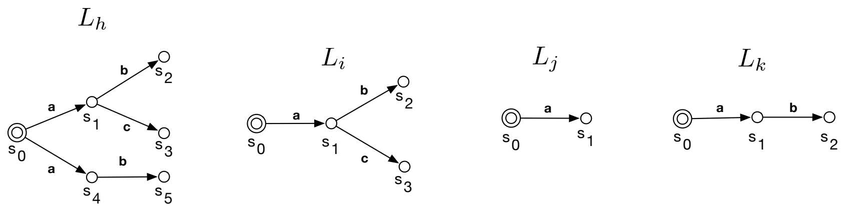


Fig. 4. Example where logistics automaton L_i is included in L_h but L_h is not included in L_i; and logistics automaton L_j is not included in logistics automaton L_k.

algebraic properties. In this fashion we define an algebra on logistics automata to systematically relate their languages, their state-space sizes and their optimization-space sizes.

4.1 Equivalence and inclusion relations

We start by defining equivalence and inclusion relations on logistics automata that capture both behavioral and structural aspects. The behavioral aspect relates the languages of the automata, while the structural aspect relates their state-space sizes.

Definition 10 (Equivalence). *Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\longrightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\longrightarrow}_2, \mathcal{S}_{0_2} \rangle$ be logistics automata. Then L_1 and L_2 are equivalent, written $L_1 \approx L_2$, if and only if $\mathcal{A}ct_1 = \mathcal{A}ct_2$ and either i) $\mathcal{S}_1 = \emptyset$ and $\mathcal{S}_2 = \emptyset$ or ii) $\mathcal{S}_{0_1} = \{s_{0_1}\}$ and $\mathcal{S}_{0_2} = \{s_{0_2}\}$ and there exists a bijective function $\mathcal{F} : \mathcal{S}_1 \longrightarrow \mathcal{S}_2$ satisfying:*

(1) *$\mathcal{F}(s_{0_1}) = s_{0_2}$;*
(2) *For all $s, s' \in \mathcal{S}_1$ and $a \in \mathcal{A}ct_1$ $s \xrightarrow{a} s'$ if and only if $F(s) \xrightarrow{a} F(s')$.*

Theorem 1. *$\approx$ is an equivalence relation on logistics automata.*

Equivalent logistics automata encode the same languages, as stated in the following lemma.

Lemma 9. *Let L_1 and L_2 be equivalent logistics automata. Then $\mathcal{L}(L_1) = \mathcal{L}(L_2)$.*

In correspondence to the equivalence relation we define a partial order (inclusion) relation on logistics automata.

Definition 11 (Inclusion). *Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\longrightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\longrightarrow}_2, \mathcal{S}_{0_2} \rangle$ 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 $s_2 \dot{\longrightarrow}_2$ then $s_1 \dot{\longrightarrow}_1$.*

Theorem 2. *$\sqsubseteq$ is a partial order relation on logistics automata. In particular $L_1 \approx L_2$ if and only if $L_1 \sqsubseteq L_2$ and $L_2 \sqsubseteq L_1$.*

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.

Lemma 10. *Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\longrightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\longrightarrow}_2, \mathcal{S}_{0_2} \rangle$ be logistics automata such that $L_1 \sqsubseteq L_2$. Then $\#\mathcal{S}_1 \leq \#\mathcal{S}_2$, where operator # counts the number of states.*

Example 4. *Consider logistics automata L_h and L_i depicted in Figure 4. It is easy to check that L_i is included in L_h, but L_h is not included in L_i. L_h and L_i are thus not equivalent, even though they have the same language. Note that our inclusion relation is stronger than Milner's [14] simulation relation $\prec$ for which both $L_h \prec L_i$ and $L_i \prec L_h$ hold. $\prec$ is a pre-order but not a partial order in the sense that $L_i \nsim L_h$, where $\sim$ denotes Milner's strong bisimulation. The essential property that makes $\sqsubseteq$ into a partial order is the property of injectivity (see Definition 11). Consider logistics automata L_j and L_k depicted in Figure 4. Even though $L_j \prec L_k$ holds, we have $L_j \not\sqsubseteq L_k$. This is because the inclusion relation forces (by condition 3. of Definition 11) all activity sequences of $\mathcal{L}(L_j)$ to be included in $\mathcal{L}(L_k)$.*

4.2 Algebraic properties

We defined an algebra on (logistics and constraint) automata with composition ($\odot$), constraining ($\upharpoonright$), MaxPlus and Tree operators, together with a partial-order ($\sqsubseteq$, $\subseteq$) and equivalence relations ($\approx$, $=$). The algebra satisfies several properties of which the importance is demonstrated in Section 6.

Theorem 3. *Let L_1, L_2 and L_3 be logistics automata and let C_1 and C_2 be constraints. Then:*

1. *$L_1 \odot L_2 \approx L_2 \odot L_1$ (commutativity $\odot$).*
2. *$(L_1 \odot L_2) \odot L_3 \approx L_1 \odot (L_2 \odot L_3)$ (associativity $\odot$).*
3. *$(L_1 \upharpoonright C_1) \upharpoonright C_2 \approx (L_1 \upharpoonright C_2) \upharpoonright C_1$ if C_1 and C_2 are constraints on L_1 (commutativity $\upharpoonright$).*
4. *$(L_1 \odot L_2) \upharpoonright C_1 \approx (L_1 \upharpoonright C_1) \odot (L_2 \upharpoonright C_1)$ if C_1 is a constraint on both L_1 and L_2 (distributivity $\upharpoonright$).*
5. *$L_1 \sqsubseteq L_2$ implies $L_1 \odot L \sqsubseteq L_2 \odot L$ (substitutivity $\sqsubseteq$ under $\odot$).*
6. *$L_1 \approx L_2$ implies $L_1 \odot L \approx L_2 \odot L$ (substitutivity $\approx$ under $\odot$).*
7. *$\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ implies $\mathcal{L}(L_1 \odot L_3) \subseteq \mathcal{L}(L_2 \odot L_3)$ (substitutivity $\subseteq$ under $\odot$).*
8. *$\mathcal{L}(L_1) = \mathcal{L}(L_2)$ implies $\mathcal{L}(L_1 \odot L_3) = \mathcal{L}(L_2 \odot L_3)$ (substitutivity $=$ under $\odot$).*
9. *$L_1 \sqsubseteq L_2$ implies $L_1 \upharpoonright C_1 \sqsubseteq L_2 \upharpoonright C_1$ if C_1 is a constraint on L_1 and L_2 (substitutivity $\sqsubseteq$ under $\upharpoonright$).*
10. *$L_1 \approx L_2$ implies $L_1 \upharpoonright C_1 \approx L_2 \upharpoonright C_1$ if C_1 is a constraint on L_1 and L_2 (substitutivity $\approx$ under $\upharpoonright$).*
11. *$\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ implies $\mathcal{L}(L_1 \upharpoonright C_1) \subseteq \mathcal{L}(L_2 \upharpoonright C_1)$ if C_1 is a constraint on L_1 and L_2 (substitutivity $\subseteq$ under $\upharpoonright$).*
12. *$\mathcal{L}(L_1) = \mathcal{L}(L_2)$ implies $\mathcal{L}(L_1 \upharpoonright C_1) = \mathcal{L}(L_2 \upharpoonright C_1)$ if C_1 is a constraint on L_1 and L_2 (substitutivity $=$ under $\upharpoonright$).*
13. *$L_1 \sqsubseteq L_2$ implies $MaxPlus(L_1) \sqsubseteq MaxPlus(L_2)$ (substitutivity $\sqsubseteq$ under MaxPlus).*
14. *$L_1 \approx L_2$ implies $MaxPlus(L_1) \approx MaxPlus(L_2)$ (substitutivity $\approx$ under MaxPlus).*
15. *$L_1 \sqsubseteq L_2$ implies $Tree(L_1) \sqsubseteq Tree(L_2)$ (substitutivity $\sqsubseteq$ under Tree).*
16. *$L_1 \approx L_2$ implies $Tree(L_1) \approx Tree(L_2)$ (substitutivity $\approx$ under Tree).*
17. *$\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ implies $\mathcal{L}(MaxPlus(L_1)) \subseteq \mathcal{L}(MaxPlus(L_2))$ (substitutivity $\subseteq$ under MaxPlus).*
18. *$\mathcal{L}(L_1) = \mathcal{L}(L_2)$ implies $\mathcal{L}(MaxPlus(L_1)) = \mathcal{L}(MaxPlus(L_2))$ (substitutivity $=$ under MaxPlus).*
19. *$\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ implies $\mathcal{L}(Tree(L_1)) \subseteq \mathcal{L}(Tree(L_2))$ (substitutivity $\subseteq$ under Tree).*
20. *$\mathcal{L}(L_1) = \mathcal{L}(L_2)$ implies $\mathcal{L}(Tree(L_1)) = \mathcal{L}(Tree(L_2))$ (substitutivity $=$ under Tree).*
21. *$L_1 \sqsubseteq MaxPlus(L_1) \sqsubseteq Tree(L_1)$ (optimization-space bounds).*
22. *$\mathcal{L}(L_1) = \mathcal{L}(MaxPlus(L_1)) = \mathcal{L}(Tree(L_1))$ (MaxPlus and Tree preserve language).*
23. *$L_1 \sqsubseteq L_2$ implies $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ ($\sqsubseteq$ implies $\subseteq$).*
24. *$L_1 \approx L_2$ implies $\mathcal{L}(L_1) = \mathcal{L}(L_2)$ ($\approx$ implies $=$).*

5 OPTIMIZATION-SPACE REDUCTION

In the previous section we introduce an algebra on logistics automata that is able to relate their languages and state-space sizes. In this section we discuss how to apply this algebra to reduce [3] the optimization-space by constraining logistics automata. For this purpose we establish sufficient

[3]With state-space reduction we mean that the state-space will not grow.

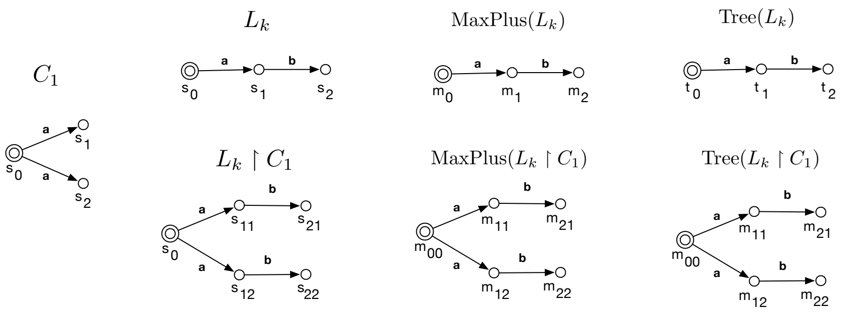

C_1
L_k
$\mathrm{MaxPlus}(L_k)$
$\mathrm{Tree}(L_k)$
$L_k \upharpoonright C_1$
$\mathrm{MaxPlus}(L_k \upharpoonright C_1)$
$\mathrm{Tree}(L_k \upharpoonright C_1)$
a
b
s0
s1
s2
s11
s12
s21
s22
m0
m1
m2
m00
m11
m12
m21
m22
t0
t1
t2

Fig. 5. The constraining of L_k with a non-deterministic constraint C_1 results in a constrained automaton $L_k \upharpoonright C$ which is not included in L_k. The same is true for their $(max, +)$ and tree counterparts.

conditions on L and C so that $L \upharpoonright C \sqsubseteq L$. First recall from Lemma 3 that $\mathcal{L}(L \upharpoonright C) \subseteq \mathcal{L}(L)$. This does not imply however that $L \upharpoonright C$ is included in L. In general even $\mathrm{MaxPlus}(L \upharpoonright C)$ is not included in $\mathrm{MaxPlus}(L)$ and $\mathrm{Tree}(L \upharpoonright C)$ is not included in $\mathrm{Tree}(L)$. This implies that constraining does not in general lead to optimization-space reduction, not even a reduction of the worst-case optimization-space. This is demonstrated in Example 5.

Example 5. *Consider logistics automaton L_k, constraint automaton C_1 and constrainted automaton $L_k \upharpoonright C_1$ depicted in Figure 5. Obviously $L_k \upharpoonright C_1 \not\sqsubseteq L_k$, MaxPlus$(L_k \upharpoonright C) \not\sqsubseteq$ MaxPlus(L_k) and Tree$(L_k \upharpoonright C) \not\sqsubseteq$ Tree(L_k). This is caused by constraint C_1 which is non-deterministic.*

Example 5 shows that non-deterministic constraints can increase the worst-case optimization space (which is the state-space of the Tree automaton). On the other hand, if a constraint is deterministic this worst-case optimization space will not increase, which follows from the following lemma.

Lemma 11. *Let $L = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\longrightarrow}_1, S_{0_1} \rangle$ be a logistics automaton and $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\longrightarrow}_2, S_{0_2} \rangle$ a constraint on L. If C is deterministic, then Tree$(L \upharpoonright C) \sqsubseteq$ Tree(L).*

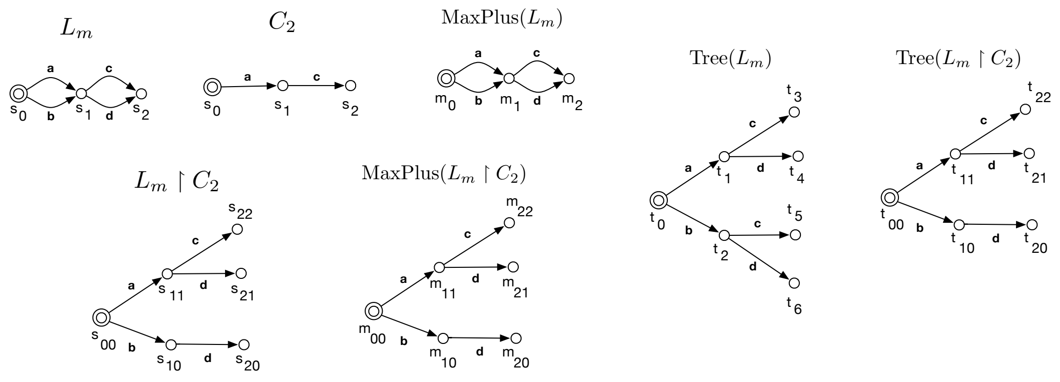
Example 6. *Consider logistics automaton L_m, constraint C_2, and constrained automaton $L_m \upharpoonright C_2$ depicted in Figure 6. Since C_2 is deterministic we have Tree$(L_m \upharpoonright C_2) \sqsubseteq$ Tree(L_m) (consistent with Lemma 11). On the other hand, we also see that $L_m \upharpoonright C_2 \not\sqsubseteq L_m$ and that MaxPlus$(L_m \upharpoonright C_2) \not\sqsubseteq$ MaxPlus(L_m). Notice in particular that constraining led to an increased size of the (max,+) state-space.*

5.1 Nonpermutation-repulsing and permutation-attracting Automata

From Lemma 11 and Example 6 we learn that deterministic constraints lead to a reduction of the worst-case optimization-space. Unfortunately, the optimization-space itself may still grow when deterministic constraints are applied. To achieve our goal in reducing the optimization-space we recall Theorem 3.13 stating that $L \upharpoonright C \sqsubseteq L$ implies $\mathrm{MaxPlus}(L \upharpoonright C) \sqsubseteq \mathrm{MaxPlus}(L)$. We thus want to make sure that $L \upharpoonright C \sqsubseteq L$. In the remainder of this section we establish sufficient conditions (on L and C) for this to be true. We start by defining the two fundamental properties of automata.

Definition 12 (Nonpermutation-repulsing). *Let $LC = \langle \mathcal{S}, \mathcal{A}ct, \dot{\longrightarrow}, \mathcal{S}_0 \rangle$ be either a logistics automaton or a constraint. Then LC is called nonpermutation-repulsing (or np-repulsing for short) if either $\mathcal{S} = \emptyset$ or $\mathcal{S} = \{s_0\}$ and the following condition holds:*

(1) *For all $s \in \mathcal{S}$ and $\underline{a}, \underline{a}' \in \mathcal{A}ct^*$, if $s_0 \xrightarrow{\underline{a}} s$ and $s_0 \xrightarrow{\underline{a}'} s$ then $\underline{a} \sim^p \underline{a}'$.*



$$M_a = M_b = \begin{bmatrix} 2 & -\infty \\ -\infty & 2 \end{bmatrix} \; M_c = M_d = \begin{bmatrix} 3 & 5 \\ -\infty & 1 \end{bmatrix}$$

Fig. 6. Example showing that in the case of a deterministic constraint C_2 the tree of the constrained automaton (Tree($L_m \upharpoonright C_2$)) is included in tree of the logistics automaton (Tree(L_m)), while this inclusion does in general neither hold for the constrained automaton nor for the MaxPlus automaton.

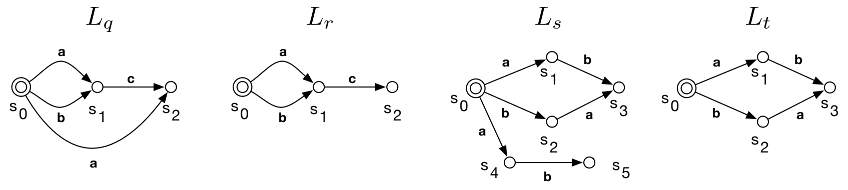


Fig. 7. Examples: L_q is neither np-repulsing nor p-attracting. L_r is p-attracting but not np-repulsing. L_s is np-repulsing but not p-attracting. L_t is both np-repulsing and p-attracting.

Here $\underline{a} \sim^p \underline{a}'$ denotes that activity sequences $\underline{a}$ and $\underline{a}'$ are permutations.

Intuitively an automaton is np-repulsing if all sequences running to a certain state are permutations of one another. This state 'repulses' sequences that are not permutations (of the sequences that lead to this state). Notice that permuting sequences do not necessarily lead to the same state. In case they do, we call the automaton permutation-attracting:

Definition 13 (Permutation-attracting). *Let $LC = \langle \mathcal{S}, \mathcal{A}ct, \dashrightarrow, \mathcal{S}_0 \rangle$ be either a logistics automaton or a constraint. Then LC is called permutation-attracting (or p-attracting for short) if either $\mathcal{S} = \emptyset$ or $\mathcal{S} = \{s_0\}$ and the following condition holds:*

(1) *For all $s, s' \in \mathcal{S}$ and $\underline{a}, \underline{a}' \in \mathcal{A}ct^*$ with $\underline{a} \sim^p \underline{a}'$, if $s_0 \xrightarrow{\underline{a}} s$ and $s_0 \xrightarrow{\underline{a}'} s'$ then $s = s'$.*

Thus all permuting sequences in the language of a p-attracting automaton, lead to the same state. This single state 'attracts' all these permutations so to say.

Example 7. *Consider the automata L_q, L_r, L_s and L_t of Figure 7. L_q is neither np-repulsing nor p-attracting. When leaving out the transition from the initial state to state s_2 we are left with automaton L_r which is p-attracting but not np-repulsing since both sequences a and b lead to state s_1. Automaton*

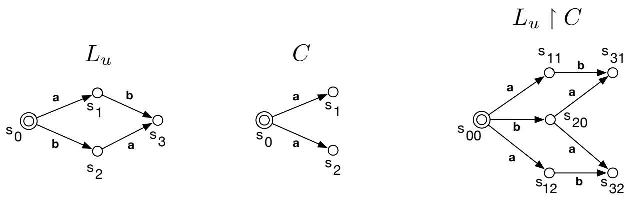


Fig. 8. Np-repulsiveness is preserved by constraining while p-attractiveness is not as shown by $L_u \upharpoonright C$.

L_s, on the other hand, is np-repulsing but not p-attracting. If states s_4 and s_5 are removed from L_s we are left with L_t which is both np-repulsing and p-attracting.

Notice that the notions of np-repulsiveness and p-attractiveness are independent. Interestingly both notions are preserved by the composition operator:

Lemma 12. *Let L_1 and L_2 be logistics automata. If L_1 and L_2 are both np-repulsing then so is $L_1 \odot L_2$. If L_1 and L_2 are both p-attracting then so is $L_1 \odot L_2$.*

We further have that np-repulsiveness is preserved by general contraining:

Lemma 13. *Let L be an np-repulsing logistics automata and let C be any constraint. Then $L \upharpoonright C$ is np-repulsing.*

Example 8. *Consider automaton L_u and constraint C depicted in Figure 8. L_u is both np-repulsing and p-attracting and C is nondeterministic. Constrained automaton $L_u \upharpoonright C$ is np-repulsing (consistent with Lemma 13), but not p-attracting.*

From the previous example we learn that p-attractiveness is not preserved by general constraining. However, if both the logistics automaton and the constraint are p-attractive, the constrained automaton is p-attractive as well, which is stated in the following lemma.

Lemma 14. *Let L be a p-attracting logistics automaton and let C be a p-attracting constraint. Then $L \upharpoonright C$ is p-attracting.*

Example 9. *Consider again automaton L_u and constraint C depicted in Figure 8. Since C is nondeterministic it is not p-attracting. Removing any of the nondeterministic branches from C would render both the constraint and the constrained automaton p-attracting.*

We now arrive at the main theorem of this paper. It establishes sufficient conditions on L and C for $L \upharpoonright C \sqsubseteq L$ to hold, thereby ensuring the constraining to imply optimization-space reduction.

Theorem 4. *Let L be an np-repulsing logistics automata and let C be a p-attracting constraint. Then $L \upharpoonright C \sqsubseteq L$.*

Example 10. *Consider the np-repulsing logistics automaton L_v and the p-attracting constraint C as shown in Figure 9. Consistent with Theorem 4 we observe that $L_v \upharpoonright C \sqsubseteq L_v$. Further from Lemmas 3.13 and 3.15 we know that both $MaxPlus(L_v \upharpoonright C) \sqsubseteq MaxPlus(L_v)$ and $Tree(L_v \upharpoonright C) \sqsubseteq Tree(L_v)$. Consequently from Lemma 10 we know that the both the (max,+) optimization space and the worst-case optimization space are reduced, which is also obvious from the Figure 9.*

The reader may have wondered about the relations between np-repulsiveness and p-attractiveness and the well-known concept of confluence. Indeed the automaton L_v depicted in Figure 9 is confluent since it has the so-called diamond property. In the remainder of this section we will show that a confluent logistics automaton is both np-repulsing and p-attracting, but not vice versa. We will also show that a confluent constraint automaton is p-attracting, but not necessarily np-repulsing. If a confluent constraint is non-recursive, it is also np-repulsing. As a consequence, our main Theorem 4 applies when both the logistics automaton and the constraint automaton are confluent. To prove this, we define the notion of confluence based on Milner's work on CCS [14] in which confluence is defined for labeled transition systems. To this end, we first define (following the definition in [14]) for two activity sequences $\underline{b}$ and $\underline{c}$, the excess of $\underline{b}$ over $\underline{c}$ which is written as $\underline{b}/\underline{c}$. We obtain $\underline{b}/\underline{c}$ by working through $\underline{b}$ from left to right deleting any activity which occurs in $\underline{c}$, taking into account the multiplicity of occurrence.

Definition 14 (Excess). *Let $\underline{b}, \underline{c} \in \mathcal{A}ct^*$ be activity sequences. Then $\underline{b}/\underline{c}$ is defined recursively upon $\underline{b}$ as:*

$$\epsilon/\underline{c} = \epsilon$$

$$(a\underline{b})/\underline{c} = \begin{cases} a(\underline{b}/\underline{c}) & \text{if } a \text{ does not occur in } \underline{c} \\ \underline{b}/(\underline{c}/a) & \text{if } a \text{ occurs in } \underline{c} \end{cases}$$

This excess operator has a number of important properties.

Lemma 15. *Let $\underline{b}, \underline{c} \in \mathcal{A}ct^*$ be activity sequences and let $a \in \mathcal{A}ct$ be an activity. Then*

(1) $\#_a(\underline{b}/\underline{c}) = max(\#_a(\underline{b}) - \#_a(\underline{c}), 0)$, *where $\#_a(\underline{b})$ denotes the number of occurrences of a in $\underline{b}$;*
(2) $\underline{b} \sim^p \underline{c}$ *if and only if* $\underline{b}/\underline{c} = \epsilon$ *and* $\underline{c}/\underline{b} = \epsilon$;
(3) $\underline{b}/\underline{c} = \underline{b}$ *if b and c have no activities in common;*
(4) $(\underline{b}/\underline{c})/(\underline{c}/\underline{b}) = \underline{b}/\underline{c}$.

We will now define the notion of confluence on logistic and constraint automata.

Definition 15 (Confluence). *Let $LC = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be either a logistics automaton or a constraint. Then LC is called confluent if either $\mathcal{S} = \emptyset$ or $\mathcal{S} = \{s_0\}$ and the following condition holds:*

- *For all $s, s_1, s_2 \in \mathcal{S}$ and $\underline{b}, \underline{c} \in \mathcal{A}ct^*$, if $s \xrightarrow{\underline{b}} s_1$ and $s \xrightarrow{\underline{c}} s_2$, then for some $s' \in \mathcal{S}$, $s_1 \xrightarrow{\underline{c}/\underline{b}} s'$ and $s_2 \xrightarrow{\underline{b}/\underline{c}} s'$.*

Every confluent logistics automaton or constraint is p-attracting which is claimed by the following lemma.

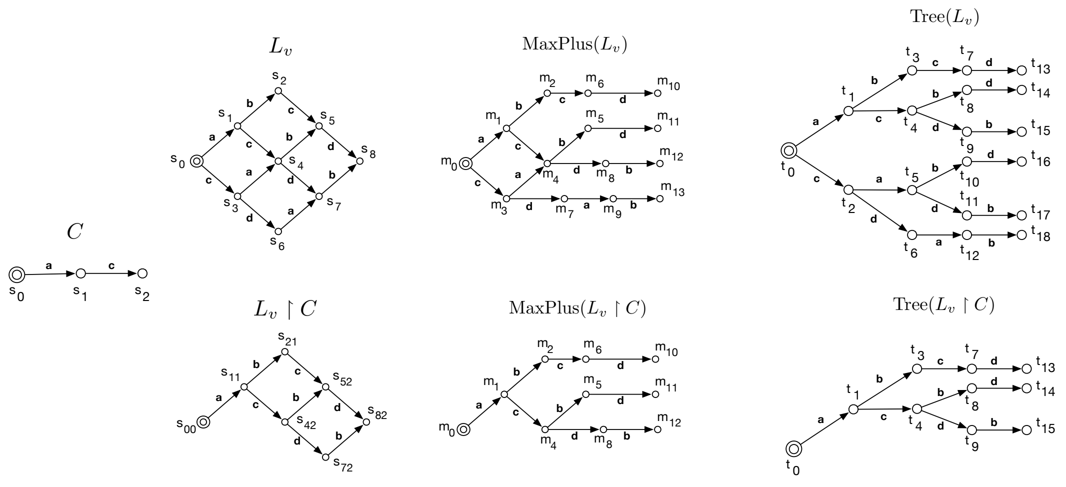
Lemma 16. *Let $LC = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a confluent logistics automaton or constraint. Then LC is p-attracting.*

With respect to np-repulsiveness a similar relation holds, but only for logistics automata. A confluent constraint is not necessarily np-repulsing, which is shown in the following example.

Example 11. *Consider constraint C depicted in Figure 10. C is confluent and p-attracting (consistent with Lemma 16). C is not np-repulsing however, since the nonpermuting sequences a and ab both lead to state s_0.*

A confluent logistics automaton, on the other hand, is always np-repulsive. This also holds for constraints that are non-recursive. This is posed in the following lemma.

Lemma 17. *Let $C = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a confluent logistics automaton or non-recursive constraint. Then C is np-repulsing.*


C
L_v
$\text{MaxPlus}(L_v)$
$\text{Tree}(L_v)$
$L_v \upharpoonright C$
$\text{MaxPlus}(L_v \upharpoonright C)$
$\text{Tree}(L_v \upharpoonright C)$

$$M_a = \begin{bmatrix} 2 & -\infty \\ -\infty & 2 \end{bmatrix} \; M_b = \begin{bmatrix} 3 & -\infty \\ 4 & 3 \end{bmatrix} \; M_c = \begin{bmatrix} 2 & -\infty \\ -\infty & 2 \end{bmatrix} \; M_d = \begin{bmatrix} 3 & 5 \\ -\infty & 3 \end{bmatrix}$$

Fig. 9. Constraining an np-repulsing automaton with a p-attracting constraint reduces the (max,+) optimization-space as well as the worst-case optimization-space.

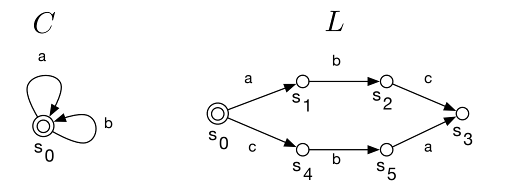


Fig. 10. A confluent constraint is not necessarily np-repulsing (C) and an np-repulsing, p-attracting logistics automaton is not necessarily confluent (L).

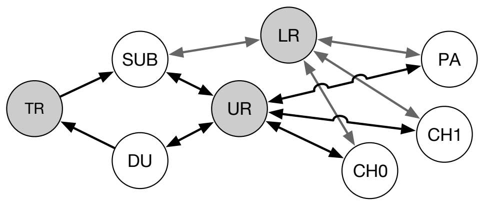


Fig. 11. Resources and possible product routes of the wafer handling controller.

From Lemmas 16 and 17 we thus know that every confluent logistics automaton and non-recursive constraint is both np-repulsing and p-attracting. The opposite is not true, which is demonstrated in the following example.

Example 12. *Consider automaton L depicted in Figure 10. L is both np-repulsing and p-attracting. But for instance since s_1 cannot make a transition with label c, L is not confluent.*

For our our main reduction Theorem 4 to apply, the logistics automaton should be np-repulsing and the constraint should be p-attracting. From From Lemma's 16 and 17 we thus know this holds in case of a confluent automaton and constraint.

6 EXPLOITING THE ALGEBRA

We defined equivalence and inclusion relations on logistics automata and their languages, and proved that these relations are substitutive under all operators. In addition we proved commutativity, associativity and distributivity properties and defined the notions of np-repulsiveness and p-attractiveness. With this we defined an algebra on logistics automata. This algebra allows us to compare logistics specifications in a modular (algebraic) way, by systematically relating their languages, their state-space and optimization-space sizes and their solutions to the BMO problem. In addition it allows the exploitation of over-specification by the systematic introduction of constraints to solve the BMO problem (for the constrained system) or BMO bounds (for the unconstrained system). Hence the algebra facilitates a specification style and approach to keep the makespan optimization problem in check. This is illustrated by the following examples.

Example 13. *Assume that we want to compare specifications* $(((L_1 \odot L_2) \restriction C_1) \odot L_3) \restriction C_2$ *and* $((L_1 \odot L_2) \restriction C_1) \odot L_3$ *where* C_2 *is a constraint on* L_1, L_2 *and* L_3 *and* L_1, L_2 *and* L_3 *are np-repulsing automata and* C_2 *is p-attracting. We can use our algebra to "massage" specification* $(((L_1 \odot L_2) \restriction C_1) \odot L_3) \restriction C_2$ *as follows:*

$$
\begin{aligned}
&(((L_1 \odot L_2) \restriction C_1) \odot L_3) \restriction C_2\\
&\approx \{\textit{Distributivity of } \restriction \textit{ (Theorem 3.4)}\}\\
&\big((L_1 \odot L_2) \restriction C_1 \restriction C_2\big) \odot (L_3 \restriction C_2)\\
&\approx \{\textit{Commutativity of } \restriction \textit{ (Theorem 3.3), substitutivity of } \approx \textit{ under } \odot \textit{ (Theorem 3.6)}\}\\
&\big((L_1 \odot L_2) \restriction C_2 \restriction C_1\big) \odot (L_3 \restriction C_2)\\
&\approx \{\textit{Distributivity of } \restriction \textit{ (Theorem 3.4), substitutivity of } \approx \textit{ under } \restriction \textit{ (Theorem 3.10)}\\
&\textit{and substitutivity of } \approx \textit{ under } \odot \textit{ (Theorem 3.6)}\}\\
&\big((L_1 \restriction C_2 \odot L_2 \restriction C_2) \restriction C_1\big) \odot (L_3 \restriction C_2)\\
&\sqsubseteq \{L_1 \restriction C_2 \sqsubseteq L_1, L_2 \restriction C_2 \sqsubseteq L_2 \textit{ and } L_3 \restriction C_2 \sqsubseteq L_3 \textit{ by Theorem 4},\\
&\textit{substitutivity of } \sqsubseteq \textit{ under } \restriction \textit{ (Theorem 3.9) and } \odot \textit{ (Theorem 3.5) and}\\
&\textit{commutativity of } \odot \textit{ (Theorem 3.1)}\}\\
&((L_1 \odot L_2) \restriction C_1) \odot L_3
\end{aligned}
$$

Hence by Theorem 3.23 $\mathcal{L}((((L_1 \odot L_2) \restriction C_1) \odot L_3) \restriction C_2) \subseteq \mathcal{L}(((L_1 \odot L_2) \restriction C_1) \odot L_3)$*. Therefore a solution to the BMO problem of* $(((L_1 \odot L_2) \restriction C_1\big) \odot L_3) \restriction C_2$ *is a suboptimal solution to the BMO problem of* $((L_1 \odot L_2) \restriction C_1) \odot L_3$ *establishing a makespan bound thereof (Lemma 5). Further by Lemma 10 the state-space of* $\big(((L_1 \odot L_2) \restriction C_1\big) \odot L_3) \restriction C_2$ *is at most as large as that of* $((L_1 \odot L_2) \restriction C_1) \odot L_3$ *and by Theorem 3.13 and Lemma 10 the optimization-space of* $(((L_1 \odot L_2) \restriction C_1\big) \odot L_3) \restriction C_2$ *is at most as large as that of* $((L_1 \odot L_2) \restriction C_1) \odot L_3$*. Since the optimization-space of the constrained specification* $(((L_1 \odot L_2) \restriction C_1\big) \odot L_3) \restriction C_2$ *is at most as large as that of the unconstrained specification* $((L_1 \odot L_2) \restriction C_1) \odot L_3$*, the effort to compute a solution is either smaller or equal to that of computing a solution of the unconstrained specification* $((L_1 \odot L_2) \restriction C_1) \odot L_3$*. Therefore, the suboptimal solution and makespan bound of the BMO problem of* $((L_1 \odot L_2) \restriction C_1) \odot L_3$ *is therefore easier to compute than the optimal solution and makespan thereof.*

Example 14. *Now assume that we want to reduce the state-space and optimization-space of* $(L_1 \restriction C_1 \odot L_2) \restriction C_2$ *where* L_2 *is an np-repulsing automaton. We can do so by applying some p-attracting constraint* C_3 *to* L_2*. Then by Theorem 4 we know that* $L_2 \restriction C_3 \sqsubseteq L_2$*. Subsequently by applying substitutivity of* $\sqsubseteq$ *under* $\odot$ *(Theorem 3.5) we have* $(L_1 \restriction C_1 \odot L_2 \restriction C_3) \sqsubseteq (L_1 \restriction C_1 \odot L_2)$*. But then by applying substitutivity of* $\sqsubseteq$ *under* $\restriction$ *(Theorem 3.9), we have* $(L_1 \restriction C_1 \odot L_2 \restriction C_3) \restriction C_2 \sqsubseteq (L_1 \restriction C_1 \odot L_2) \restriction C_2$*. Similar to Example 13 we can now compute the BMO solution to* $(L_1 \restriction C_1 \odot L_2 \restriction C_3) \restriction C_2$*. This gives a suboptimal solution to the BMO problem of* $(L_1 \restriction C_1 \odot L_2) \restriction C_2$*, but it can be computed more efficiently then the optimal solution.*

Using our algebra of logistics automata we can define a systematic approach for the batch-oriented specification and optimization of the logistics of flexible manufacturing systems as follows:

(1) For each product in a batch, we write its product flow as an np-repulsing logistics automaton. Note that this can always be done, since any logistics automaton can be written as a language-equivalent automaton which is np-repulsing. For example, any logistics automaton L can be written in its tree form $\text{Tree}(L)$ which has the same language and is np-repulsing. In some cases the product automata are actually confluent, and also np-repulsing (Lemma 17).
(2) The automaton describing a batch is then obtained via the composition of all individual product automata. By Lemma 12 we know that the resulting automaton is also np-repulsing.
(3) Each system constraint should preferably be written as a p-attracting (or even confluent) constraint automaton. This is important since by Theorem 4 and Lemma 10 the constraining of an np-repulsing automaton with a p-attracting constraint automaton leads to a reduction of the state-space and optimization-space of the logistics automaton. If a system constraint cannot be written (or conveniently written) as a p-attracting automaton step 4 (applying heuristics) is still applicable. This is because the constraining of a logistics automaton with any constraint preserves its np-repulsiveness (Lemma 13).
(4) In some cases the optimization-space is still too large to efficiently compute a solution to the BMO problem. In these cases we can introduce additional (non-essential) constraints to the specification. These should be written as p-attracting (or even confluent) constraint automata such that the optimization-space is effectively pruned, following Theorem 4 and Lemma 10.

Using this systematic approach to the specification of batch-oriented logistics one can keep the makespan optimization problem in check. Further, it allows the exploitation of over-specification by the systematic introduction of additional requirements to solve the BMO problem (for the over-constrained system) or to find BMO makespan bounds (for the initial system specification). Notice that the technique does not allow us to make a quantitative statement about the quality of the solution obtained, if the unconstrained optimization-space can not be computed.

7 CASE STUDY

This section presents a case study in which we apply our framework to specify a wafer handling controller and optimize its makespan. The wafer handling controller is one of the controllers of an ASML TWINSCAN™ lithography scanner. These are manufacturing systems responsible for the photo-lithography step in the semiconductor manufacturing process where circuit patterns are exposed onto silicon wafers. Before this expose operation can occur, wafers must be correctly conditioned (for temperature and orientation) and measured to detect defects on its surface. In the case of the TWINSCAN™ , the measure and expose operations are decoupled which allows the system to increase productivity by executing them in parallel. To achieve this, the scanner contains two chucks that can change locations to perform either the measure or the expose operation.

We consider the input of the system to be a batch of 25 wafers which is the typical number of wafers stored in a Front Opening Unified Pod (FOUP) [10]. In the next section we describe the system in terms of its resources, activities, logistics and system requirements. We omit any specification in terms of locations, layout, actions and execution times for confidentially reasons.

(**a**) Track_2_SUB	(**b**) SUB_Conditioning	(**c**) SUB_2_UR	(**d**) UR_2_PA	(**e**) PA_PreAlign
(**f**) PA_2_LR	(**g**) LR_2_CH0	(**h**) CH0_Measure	(**i**) CH0_Expose	(**j**) CH0_2_UR
(**k**) LR_2_CH1	(**l**) CH1_Measure	(**m**) CH1_Expose	(**n**) CH1_2_UR	(**o**) UR_2_DU
(**p**) CH0_M_2_E	(**q**) CH1_M_2_E	(**r**) CH0_E_2_M	(**s**) CH1_E_2_M	

Table 1. Set of activities of the wafer handling controller.

7.1 Resources and Activities

For wafer handling eight resources are utilized which are depicted in Figure 11 by annotated circles. Circles with a white background represent the production resources SUB, DU, PA, CH0 and CH1, while darkened circles represent transport resource TR, LR and UR. There are two robots which are used to transport wafers between the different processing units, the Load Robot (LR) and the Unload Robot (UR). The Storage Unit (SUB) ensures that each wafer is conditioned to a predefined temperature. The Pre-Aligner (PA) accurately aligns a wafer with respect to a reference position. The Discharge Unite (DU) is an output buffer. The Track (TR) is a system external to the system responsible for the input and output to and from the scanner. Finally, we have two chucks (CH0 and CH1) which are able to perform the scanner operations measure and expose.

Table 1 lists all the activities necessary to capture the good-weather manufacturing of a batch of wafers. In this context good-weather means that we do not account for error or test scenarios. Therefore many of the possible movement operations between resources, depicted by the directional arrows in Figure 11, are not utilized. 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_SUB is referred by letter **a**). To denote an activity that is performed on behalf of wafer *i* we attach a corresponding subscript to the activity name. For example, activity **a**_3 represents activity Track_2_SUB on behalf of wafer 3. Activity **a** captures the input operation during which a wafer is loaded into the scanner by the TR resource. Similarly, activity **o** captures the output of an exposed wafer from the system by placing it on the DU resource. Activities **b** and **e** capture the pre-processing operations concerning conditioning and pre-aligning. Activity **b** conditions the wafer temperature for an accurate exposure and activity **e** aligns the wafer to minimize the overlay error of multiple exposures. Activities **l** and **h** capture the measure operations on the CH0 and CH1 resources, respectively, and activities **m** and **i** capture the expose operations on the CH0 and CH1 resources, respectively. Resources CH0 and CH1 need to be in specific locations to perform the measure and expose operations. Activities **p, q, r** and **s** capture the movement of resources CH0 and CH1 from and to the measure and expose locations. For example activity CH0_M_2_E (**p**) captures the moving of resource CH0 from the measure location (M) to the expose location (E). Activities **c, d, f, j, g, k, n** and **o** capture the movement of wafers across the system using resources LR and UR. These are always described by an initial and destination resource. For example, activity PA_2_LR (**f**) captures the movement of a wafer from the PA resource to the LR resource.

7.2 Wafer Logistics

For every wafer we define its life-cycle (wafer product flow). Figure 12 depicts logistics automaton L_{LC_i} which captures the life-cycle of wafers *i* in a batch of wafers. This life-cycle starts with the input of a wafer into the scanner by placing it on the SUB resource. This is done by the TR resource (activity **a**_i). Once on the SUB resource, the wafer is conditioned to a pre-defined range of temperatures (activity **b**_i). After conditioning, the wafer is moved from the SUB resource to the PA resource using the UR resource as an intermediary (activities **c**_i and **d**_i). The PA resource accurately aligns the wafer to a reference (activity **e**_i). Once aligned, the wafer is picked by the LR resource from the PA (activity **f**_i). At this point the controller has a choice to either load the wafer on chuck resource CH0 or on chuck resource CH1, (activity **g**_i or activity **k**_i respectively). This is visible in automata L_{LC_i} by the branching after activity **f**_i. Assume that the controller picks the CH0 resource (upper branch). In this case, the wafer is measured and then exposed using resource CH0. After measuring (activity **h**_i), resource CH0 moves from the measure location to the expose location (activity **p**_i). Once in the expose location the wafer is exposed with a certain circuit pattern (activity **i**_i). Then resource CH0 moves from the expose location to the measure location

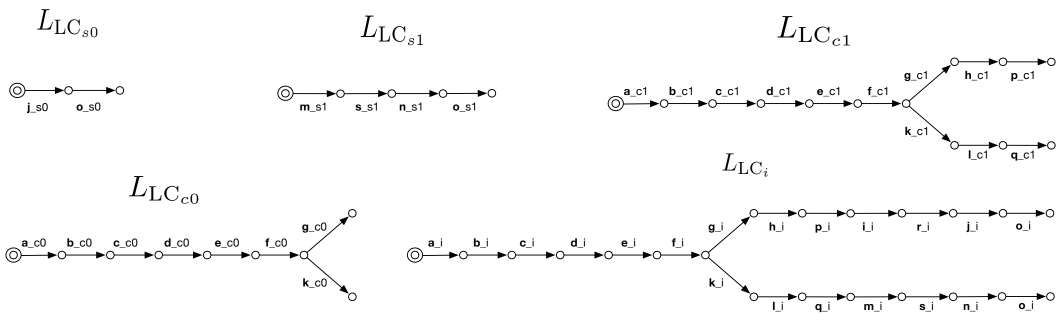


Fig. 12. Logistics automaton L_{LC_i} capturing the life-cycle for a wafer i of a batch of wafers and logistics automata $L_{\mathrm{LC}_{s0}}$, $L_{\mathrm{LC}_{s1}}$, $L_{\mathrm{LC}_{c1}}$ and $L_{\mathrm{LC}_{c0}}$ capturing the life-cycle of dummy wafers s0, s1, c1 and c0 respectively.

(activity **r**_i). Once returned at the measure location, the wafer is moved from the CH0 resource to the UR resource and finally placed on the DU resource to be outputted by the TR resource (activities **j**_i and **o**_i, respectively). In case the controller picks the CH1 the product flow remains similar, however, the activities for the CH1 resource would be executed **k**_i, **l**_i, **q**_i, **m**_i, **s**_i and **n**_i.

Whenever the system is not in production, dummy wafers are loaded on the chuck resources for safety reasons. For this case-study we take this into account by modeling four dummy wafers s0, s1, c0 and c1. Dummy wafers s0 and s1 represent the dummy wafers already in the system before the processing of a FOUP, while c0 and c1 represent the dummy wafers that should be loaded once the FOUP is processed. We assume that CH0 starts at the measure location and that CH1 starts at the expose location. This is enforced in the life-cycle of dummy wafers s0 and s1 captured by logistics automata $L_{\mathrm{LC}_{s0}}$ and $L_{\mathrm{LC}_{s1}}$ depicted in Figure 12. The life-cycle of s0 starts with the unloading of the dummy wafer from the CH0 resource to the UR resource (activity **j**_s0) and ends with its placement on the output buffer DU (activity **o**_s0). The life-cycle of s1 starts with the expose activity of dummy wafer s1 (activity **m**_s1) followed by the movement of the CH1 resource to the measure location (activity **s**_s1), its unloading to the UR resource (activity **n**_s1) and finally its placement on the output buffer DU (activity **o**_s1). The life-cycle of dummy wafers c0 and c1 follows the same flow as the life-cycle of a production wafer but ends in different final states. These are captured by logistics automata $L_{\mathrm{LC}_{c0}}$ and $L_{\mathrm{LC}_{c1}}$ depicted in Figure 12. The life-cycle of c0 ends once the dummy wafer is loaded onto either the CH0 resource or the CH1 resource (activity **g**_c0 or **k**_c0). The life-cycle of c1 ends once the dummy wafer is moved to the expose location placed either on the CH0 resource and CH1 resource (activity **p**_c1 or **q**_c1).

The batch specification of a batch of 25 wafers is given by the following logistics expression: $L_{\mathrm{LC}_{s0}} \odot L_{\mathrm{LC}_{s1}} \odot (L_{\mathrm{LC}_1} \odot L_{\mathrm{LC}_2} \odot \cdots \odot L_{\mathrm{LC}_{25}}) \odot L_{\mathrm{LC}_{c1}} \odot L_{\mathrm{LC}_{c0}}$. For brevity we will refer to this composite automaton as L_{LC}. Note that each of the life-cylce automata in Figure 12 is np-repulsing and that by Lemma 12 the composition L_{LC} is therefore also np-repulsing.

7.3 System Requirements

On top of the product flow there are certain functional requirements on the manufacturing of a batch of wafers. These are listed and explained below:

(1) *Products shall enter and leave the system in a First-In-First-Out (FIFO) order.*
(2) *There shall only be one product at a time in each resources (unary capacity).*
(3) *Wafers shall not collide (i.e. products shall not be be placed on a occupied resource).*
(4) *Chucks must swap positions in between every measure and expose activity (Swap).*
(5) *Chucks must be loaded and unloaded at the measure location.*

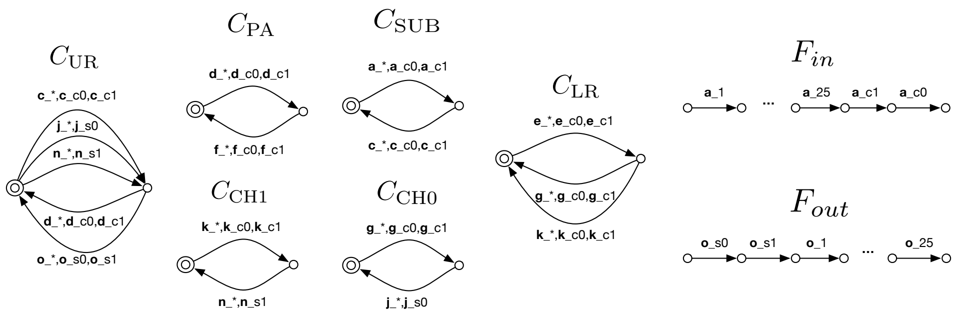


Fig. 13. Constraint automata $C_{\mathrm{SUB}}, C_{\mathrm{UR}}, C_{\mathrm{LR}}, C_{\mathrm{PA}}, C_{\mathrm{CH0}}$ and C_{CH1} capturing capacity requirements for the wafer handling resources and constraint automata F_{in} and F_{out} capturing the FIFO input and output ordering

Requirement 1 is captured by constraint automata F_{in} and F_{out} depicted in Figure 13. This is done by ordering input and output activities **a** and **o** for all wafers in the FOUP and dummy wafers s0, s1, c0 and c1. Requirements 2 and 3 are satisfied by enforcing that activities that place a wafer on a resource can only occur after a corresponding activity that picks a wafer from that resource and vice-versa. This is captured by p-attracting constraint automata $C_{\mathrm{SUB}}, C_{\mathrm{UR}}, C_{\mathrm{LR}}, C_{\mathrm{PA}}, C_{\mathrm{CH0}}$ and C_{CH1} depicted in Figure 13. For example, consider automata C_{SUB}. The initial state represents the resource to be empty and the other state represents the resource to be occupied. To avoid cluttering the automata figures we write **act**_$*$ to represent 25 different transitions with labels **act**_1, $\cdots$, **act**_25 corresponding to the 25 production wafers in a FOUP (where **act** denotes the name of an activity). If the resource is empty, activities **a**_$*$, **a**_c1 and **a**_c0 are enabled since these imply the placing of wafer on the SUB. If an activity **a**_$*$, **a**_c1 or **a**_c0 is executed, then it is enforced that another activity **a**_$*$, **a**_c1 or **a**_c0 can only occur if an activity **c**_$*$, **c**_c1 or **c**_c0 occurs in between.

Requirements 4 and 5 are partially enforced by the logistics product flow L_{LC_i} and partially by p-attracting constraint automata C_{Swap}, $C_{\mathrm{UnloadLoad}_0}$ and $C_{\mathrm{UnloadLoad}_1}$ depicted in Figure 14. Constraint automaton C_{Swap} enforces that after every execution of the measure and expose activities, chuck resources CH0 and CH1 swap positions. The initial state of C_{Swap} considers two initial situations: 1) CH0 is at the measure location and CH1 is at the expose location or 2) CH1 is at the measure location and CH0 is at the expose location. Situation 1) is captured by the lower branches and 2) by the top branches. After executing the corresponding measure and expose operation the chuck resources CH1 and CH0 swap locations. In situation 1) CH0 moves from measure to expose (activities **p**_* and **p**_c1) and CH1 from expose to measure (activities **s**_* and **s**_s1) and in 2) CH1 moves from measure to expose (activities **q**_* or **q**_c1) and CH0 from expose to measure (activities **r**_*). This flow is then repeated for each of the initial situations captured by cases 1) and 2). P-attracting constraint automata $C_{\mathrm{UnloadLoad}_0}$ and $C_{\mathrm{UnloadLoad}_1}$ enforce that the loading and unloading of a chuck to and from resources CH0 and CH1, respectively, only occurs if the respective chuck resource is at the measure location. In the case of resource CH0 (captured by $C_{\mathrm{UnloadLoad}_0}$), its starting position is at the measure location and therefore the initial state enables both the loading (activities **n**_* or **n**_s1) and the unloading (activities **k**_*, **k**_c0 or **k**_c1) of a wafer. If the wafer is unloaded the automaton moves to a state where it still enables the loading of another wafer. Once a wafer is loaded onto the resource then the automata moves to a state where it first requires that the chuck resource moves again to the measure location before allowing it to load or unload a wafer again. The flow is similar for CH1 with the difference that the CH1 resource starts at the expose location.

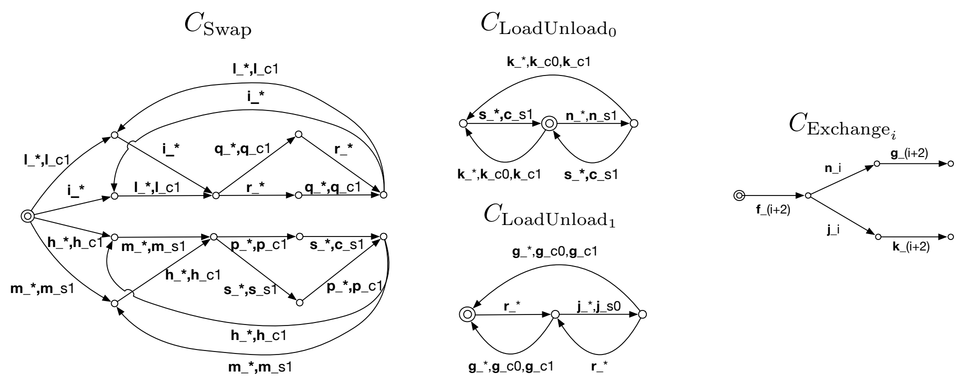


Fig. 14. Constraint C_{Swap} capturing the exchange of chuck positions after measure and expose operations; constraint $C_{\text{UnloadLoad}_0}$ and $C_{\text{UnloadLoad}_1}$ enforce that the load/unload of wafers occurs at the measure location and constraint C_{Exchange} captures a specific order of the exchange of wafers from and to CH0 and CH1.

Table 2. Size of the state-space and optimization-space, and computed makespan for a FOUP (25 wafers and 4 dummy wafers).

	State-Space		Optimization-Space		
Model	N. States	N. Transitions	N. States	N. Transitions	Makespan (s)
Logistics	-	-	-	-	-
+Capacity	-	-	-	-	-
+FIFO	65823	216611	-	-	-
+Swap	13630	39738	772806	2200049	331.7
+Exchange	9778	28225	264854	737499	331.7

The specification of a batch of 25 wafers is now obtained by constraining logistics automaton L_{LC} with all of the (p-attracting) constraints described above: $L_{\text{LC}} \upharpoonright C_{\text{COND}} \upharpoonright C_{\text{UR}} \upharpoonright C_{\text{LR}} \upharpoonright C_{\text{PA}} \upharpoonright C_{\text{CH0}} \upharpoonright C_{\text{CH1}} \upharpoonright F_{in} \upharpoonright F_{out} \upharpoonright C_{\text{Swap}} \upharpoonright C_{\text{UnloadLoad}_1} \upharpoonright C_{\text{UnloadLoad}_1}$.

7.4 Wafer Logistics Optimization

Now that we have introduce the wafer handling controller its logistics requirements and system constraints we compute the optimization-space and the solution to the BMO for a batch of 25 wafers (including the 4 dummy wafers $s0$, $s1$, $c0$ and $c1$). The values presented in this section are computed in a system with an Intel i7 920@2.67Ghz with 8 cores and 32GB of RAM memory. We use the CIF tool [18] to compute the state-spaces of logistics automata and the algorithms of [6, 19] to compute the corresponding optimization-spaces and optimal activity sequences.

Table 2 shows the results in terms of the number of states (N. States) and transitions (N. Transitions) of the state-space and optimization-space as well as the obtained minimal makespan (Makespan) for the processing of a FOUP by the wafer handler. In case we are not able to compute the state-spaces this is noted with a (-) symbol. The values for the state-space and optimization-space sizes of L_{LC} are shown in the first row (Logistics) of Table 2. Each additional row of the table represents the cumulative addition of another requirement (indicated by the + symbol). Where +Capacity refers to automaton $(L_{\text{LC}}) \upharpoonright C_{\text{COND}} \upharpoonright C_{\text{UR}} \upharpoonright C_{\text{LR}} \upharpoonright C_{\text{PA}} \upharpoonright C_{\text{CH0}} \upharpoonright C_{\text{CH1}}$, +FIFO to automaton $(L_{\text{LC}} \upharpoonright C_{\text{COND}} \upharpoonright C_{\text{UR}} \upharpoonright C_{\text{LR}} \upharpoonright C_{\text{PA}} \upharpoonright C_{\text{CH0}} \upharpoonright C_{\text{CH1}}) \upharpoonright F_{in} \upharpoonright F_{out}$ and +Swap to automaton

$(L_{\mathrm{LC}} \upharpoonright C_{\mathrm{COND}} \upharpoonright C_{\mathrm{UR}} \upharpoonright C_{\mathrm{LR}} \upharpoonright C_{\mathrm{PA}} \upharpoonright C_{\mathrm{CH0}} \upharpoonright C_{\mathrm{CH1}} \upharpoonright F_{in} \upharpoonright F_{out}) \upharpoonright C_{\mathrm{Swap}} \upharpoonright C_{\mathrm{UnloadLoad}_1} \upharpoonright C_{\mathrm{UnloadLoad}_1}$. We observe that the cumulative constraining of the system with Capacity, FIFO and Swap requirements systematically reduces the size of the state-space and optimization-space until the point that the optimization-space can be explicitly constructed and a solution to the BMO problem can be computed, yielding a minimal makespan of 331.7 seconds. For the specifications which we are able to compute the logistics automaton, a solution to the BMO problem is computed within 2 minutes. Even though it is sufficient to apply all system constraints to compute the makespan optimal activity sequence, we demonstrate the effectiveness of the heuristics approach by further pruning the state-space and optimization-space. Since the Capacity, FIFO and Swap requirements are p-attracting, we know that their application to L_{LC} resulted in an np-repulsing automaton. The optimization space can thus be pruned further by applying additional p-attracting automata. To illustrate this we will use over-specification by subsequently formalizing a non-essential requirement.

Each wafer must eventually be placed on either the CH0 or CH1 resource of the expose stage for the measure and expose activities to take place. This is realized by a combination of activities **j**_i and **g**_i or **n**_i and **k**_i. The process of loading and unloading a wafer to and from CH0 and CH1 always takes place from the measure location. Therefore, chucks CH0 and CH1 must always move first to the measure location before exchanging an exposed wafer for a new wafer. Further, to maximize productivity both chucks should be utilized in an effort to parallelize the measure and expose activities. For these reasons an exchange should always consider an exposed wafer i and an non-exposed wafer $i + 2$. Due to this, we figured that a proper heuristic is defined by p-attracting constraint automaton C_{Exchange_i} (depicted in Figure 14) which enforces a specific order between unload of wafer i from either CH0 or CH1 and the load of wafer $i + 2$ to the corresponding chuck resource. This constraint is instantiated for each wafer i where $1 \leq i \leq 25$. For the last two dummy wafers this requirement is not enforced. The results are shown in Table 2 in the +Exchange row. Because we could already compute the optimal solution without over-specification, we can compare quantitatively the impact of the additional constraint: it reduces the optimization-space by more than 60%. Note that the optimal result is preserved even though the system is further constrained with over-specification in this case.

8 RELATED WORK

To the best of our knowledge, this is the first work in which an algebraic framework is developed to systematically reason about state-space sizes. This work combines the ingredients from the modular constraint-oriented approach of the LOTOS framework [8] and of Supervisory Control Theory (SCT) [16] developed in [19] and applied in [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) [14], 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 work in which the inclusion relation $\sqsubseteq$ (to compare transitions systems in both a behavioral and in a structural way) is established, together with its algebraic properties.

It is important to mention that for some instances of the BMO problem other heuristics-based and approximation techniques could be applicable. For example by using heuristics from different job-shop [2] or vehicle routing [7] problems.

In addition to our approach, some of these solutions include deadlines and due dates [21–23] and other types of timing constraints [3, 4]. However, they do not focus on expressing or guaranteeing the satisfaction functional requirements. Moreover, the application of such techniques and algorithms is quite dependent on the actual instance of the problem we might be trying to solve. In [2] a survey of different solutions to diverse flexible job-shop formulation is carried out, considering systems with multiple operation assignments, multiple transport routing choices, resource sharing, setup times and other system requirements. We conjecture that our work can specify and optimize all of these variants, except those dealing with timing constraints (e.g. deadlines and due dates). Notice that a direct comparison of these optimization approaches to the design framework presented would not be a fair, since it would focus solely on the makespan computation and not on other features such as the specification methodology and the algebraic framework or guaranteeing the satisfaction of functional requirements.

9 CONCLUSIONS

In this paper we extended the framework of [6, 19] by introducing an algebra on logistics automata. This algebra allows us to compare logistics specifications in a modular (algebraic) way, by relating their languages, their state-space and optimization-space sizes and their solutions to the BMO problem. To this end we defined equivalence and inclusion relations on logistics automata and their languages and established important algebraic properties for the composition ($\odot$), constraining ($\upharpoonleft$), MaxPlus and Tree operators. We showed that our inclusion relation is a partial-order relation and that if a logistics automaton is included in another, the state-space of the former never exceeds that of the later. Further, we showed that a logistics automaton is always included in its corresponding (max,+) automaton.

We also established sufficient conditions to ensure that the constraining of a logistics automaton leads to the pruning of its state-space. To this end, we defined the notions of np-repulsiveness and p-attractiveness and showed: 1) that np-repulsiveness and p-attractiveness are preserved under the composition operator; 2) that np-repulsiveness is preserved by general constraining and 3) that the constraining of an np-repulsing logistics automaton with a p-attracting constraint automaton effectively reduces the size of the state-space and optimization-space of the logistics automaton. Further we related the notions of np-repulsiveness and p-attractiveness to the notion of confluence and showed that: 1) a confluent logistics automaton is both np-repulsing and p-attracting, but not vice versa; 2) a confluent constraint automaton is p-attracting, but not necessarily np-repulsing.

Using the algebra of logistics automata and its properties we defined a method for the batch-oriented specification of the logistics of flexible manufacturing systems. In essence, logistics requirements should be written as np-repulsing logistics automata and constraints as p-attracting constraint automata. Satisfying these conditions ensures that constraining the logistics automaton effectively reduces the size of its state-space and optimization-space. Furthermore, it also allows the exploitation of over-specification by the systematic introduction of additional requirements to solve the BMO problem (for the over-constrained system) or to find BMO makespan bounds (for the initial non over-constrained system specification). This method facilitates a specification style and optimization approach to keep the makespan optimization problem in check.

We demonstrated the applicability of the algebra and of the specification method by solving the BMO problem of an industrial case study of a wafer handling controller as well as to the xCPS system in [11]. Further we formalized a non-essential requirement of the wafer handler as a p-attracting constraint automaton and showed that its addition to the specification results in the reduction of the optimization-space of 60% compared with the original state-space and for which the optimal result was preserved.

There are a number of research directions to extend this work. An interesting direction would be perform state-space and optimization-space expansion in tandem with makespan optimization. Such an on-the-fly approach would avoid the explicit construction of the state-space and optimization-space of a logistics automaton. Furthermore, partial-order reduction techniques, such as the one proposed in [20], could be applied to avoid the exploration of redundant sequences. Another interesting research direction would be the generalization of this algebra and its application to finite-state automata. Moreover, the specification of latency requirements has not been explored in the work presented, which constitutes an important research direction to allow a broader class of flexible manufacturing systems to be specified. Finally, more case studies (e.g. in the printing domain) would help to investigate the limitations of the approach and to generalize it.

REFERENCES

[1] Marianne Akian, Ravindra Bapat, and Stephane Gaubert. 2006. Max-Plus Algebra. *Discrete Mathematics and Its Applications* 39 (11 2006). https://doi.org/10.1201/9781420010572.ch25

[2] Chaudhry Imran Ali and Khan Abid Ali. 2016. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research* 23, 3 (2016), 551–591.

[3] Ali Allahverdi. 2016. A survey of scheduling problems with no-wait in process. *European Journal of Operational Research* 255, 3 (2016), 665 – 686.

[4] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. 2008. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research* 187, 3 (2008), 985 – 1032.

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

[6] J. Bastos, J.P.M. Voeten, S. Stuijk, H. Corporaal, and R.R.H. Schiffelers. 2018. Exploiting Specification Modularity to Prune the Optimization-Space of Manufacturing Systems. *SCOPES 18.*

[7] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. 2016. The vehicle routing problem: State of the art classification and review. *Computers and Industrial Engineering* 99 (2016), 300 – 313.

[8] Ed Brinksma. 1990. Constraint-oriented specification in a constructive formal description technique. (1990).

[9] R. de Groote, J. Kuper, H. Broersma, and G. J. M. Smit. 2012. Max-Plus Algebraic Throughput Analysis of Synchronous Dataflow Graphs. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. 29–38.

[10] Robert Doering and Yoshio Nishi. 2007. *Handbook of Semiconductor Manufacturing Technology*. Vol. 32.

[11] Twan Basten et al. 2020. *Scenarios in the Design of Flexible Manufacturing Systems*. Springer International Publishing, Cham, 181–224. https://doi.org/10.1007/978-3-030-20343-6_9

[12] S.T.J. Forschelen, J.M. Mortel - Fronczak, van de, Rong Su, and J.E. Rooda. 2012. Application of supervisory control theory to theme park vehicles. *Discrete Event Dynamic Systems* 22, 4 (2012), 511–540.

[13] Bernd Heidergott, Geert Jan Olsder, and Jacob van der Woude. 2006. *Max Plus at Work: Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications*. Princeton University Press.

[14] Robin Milner. 1983. Calculi for synchrony and asynchrony. *Theoretical Computer Science* 25, 3 (1983), 267 – 310.

[15] J.P. Nogueira Bastos. 2018. *Modular specification and design exploration for flexible manufacturing systems*. Ph.D. Dissertation. Department of Electrical Engineering. Proefschrift.

[16] Peter J Ramadge and W Murray Wonham. 1987. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization* 25, 1 (1987), 206–230.

[17] Ofira Shmueli, Nava Pliskin, and Lior Fink. 2015. Explaining over-requirement in software development projects: An experimental investigation of behavioral effects. *International Journal of Project Management* 33, 2 (2015), 380 – 394.

[18] 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*.

[19] 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*.

[20] Bram van der Sanden, Marc Geilen, Michel A. Reniers, and Twan Basten. 2018. Partial-Order Reduction for Performance Analysis of Max-Plus Timed Systems. In *18th International Conference on Application of Concurrency to System Design*.

[21] J. van Pinxten, M. Geilen, T. Basten, U. Waqas, and L. Somers. 2016. Online heuristic for the Multi-Objective Generalized traveling salesman problem. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 822–825.

[22] Joost van Pinxten, Umar Waqas, Marc Geilen, Twan Basten, and Lou Somers. 2017. Online Scheduling of 2-Re-entrant Flexible Manufacturing Systems. *ACM Transactions on Embedded Computing Systems* 16, 5s, Article 160 (Sept. 2017).

[23] U. Waqas. 2017. *Scheduling and variation-aware design of self-re-entrant flowshops*. Ph.D. Dissertation. Technische Universiteit Eindhoven.