Modular specification and design exploration for flexible manufacturing systems

Modular Specification and Design Exploration for Flexible Manufacturing Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op maandag 03 december 2018 om 13:30 uur

door

João Pedro Nogueira Bastos

geboren te Porto, Portugal.

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr.ir. A.B. Smolders
1e promotor:	prof.dr.ir. J.P.M. Voeten
2e promotor:	prof.dr. H. Corporaal
copromotor(en):	dr.ir. S. Stuijk
leden:	prof.dr. J.J.M. Hooman (Radboud Universiteit Nijmegen)
	dr.ir. M.A. Reniers
	prof.dr.sc. S. Chakraborty (Technische Universität München)
adviseur(s):	dr.ir. R.R.H. Schiffelers

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

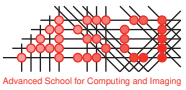
Modular Specification and Design Exploration for Flexible Manufacturing Systems

João Bastos

Doctoral committee:

prof.dr.ir. A.B. Smolders	Eindhoven University of Technology, chairman
prof.dr.ir. J.P.M. Voeten	Eindhoven University of Technology, 1st promotor
prof.dr. H. Corporaal	Eindhoven University of Technology, 2nd promotor
dr.ir. S. Stuijk	Eindhoven University of Technology, copromotor
prof.dr. J.J.M. Hooman	Radboud University Nijmegen, ESI (TNO)
dr.ir. M.A. Reniers	Eindhoven University of Technology
prof.dr.sc. S. Chakraborty	Technical University of Munich
dr.ir. R.R.H. Schiffelers	Eindhoven University of Technology, ASML

This work is part of the research program high-performance control for nano-precision systems with project number 12964, which is (partly) funded by NWO (Netherlands Organization for Scientific Research).



This work was carried out in the ASCI graduate school.
ASCI dissertation series number is 400.

Cover design: Immy Stege
Printed by: ProefschriftMaken | www.proefschriftmaken.nl

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-94-6380-091-4

Summary

Manufacturing systems are cyber-physical systems which perform operations on batches of products. Production and transport resources (the physical part) carry out operations while a logistics controller (the cyber part) assigns these operations to resources and determines their order. These systems are flexible when they allow for the manufacturing of custom and multiple product types. This is achieved by adding system functionality in the form of multiple product routes within the system. Industrial roadmaps demand a continuous improvement in both system performance and production flexibility. In order to meet this demand with minimum impact in cost, system resources are shared among different manufacturing operations. The increase in flexibility leads to involved functional system specifications which do not suit monolithic approaches (i.e. not modular). Moreover, multiple product routing options and also resource sharing increase the complexity of finding performance optimal operation assignments and orderings. As a consequence, the design of functionally correct and makespan optimal logistics becomes a difficult problem for which new design approaches are needed. This thesis incorporates several contributions in the form of a novel design framework to address these challenges. The framework relies on a formal modelling approach which is expressive enough to deal with the dynamism arising from system flexibility combined with analytical computation of makespan-optimal logistical decisions. To tackle functional complexity, system requirements are modularly specified and functionally correct logistics are obtained through automated synthesis techniques.

The thesis introduces a formal modeling approach for the specification of both functional and temporal aspects of flexible manufacturing systems. The chosen abstractions and modeling expressivity are inspired by observations of the current state-of-practice. Its novelty lies in the separation of concerns between functional and temporal aspects of the system, 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 composed of multiple peripheral actions and dependencies among them. The temporal behavior of each activity is characterized using (max,+) algebra. Functional requirements on the valid choices and orderings of activities are captured as a set of modular finite state automata. From these automata a functionally correct logistics state-space is synthesized. Any activity sequence in the logistics state-space corresponds to a correct and complete manufacturing of a batch of products. For performance analysis, a timed state-space is obtained by means of a (max,+) expansion of the logistics state-space. The timed state-space is explored with existing analysis techniques to obtain makespan-optimal logistical decisions in the form of an activity sequence.

The problem of finding an optimal logistics activity sequence is NP-Hard, which is proven in this thesis. To address scalability this thesis proposes a novel algebra to systematically relate state-space sizes, timed state-space sizes and makespan-optimal solutions. This algebra allows non-essential requirements to be exploited to prune the state-spaces. The approach is inspired from the common practice of over-specification in the design of industrial manufacturing systems. In this context, over-specification is used as a natural and implicit way to deal with complexity. Examples involve: disallowing multiple mapping possibilities for an operation and the static ordering of system operations. In this thesis over-specification is formalized in terms of constraints and explicit conditions established that are proven to result in a reduced state-space.

Once a makespan-optimal activity sequence is obtained further analysis is supported in the form of bottleneck identification to determine possible candidates for improvement. These include changes in the numbers or types of resources or even in the geometrical layout of the system. The exploration relies on critical path analysis methods to identify bottlenecks in chosen activity sequences. Since the operational processes involved in manufacturing can be stochastic in nature this thesis proposes a stochastic analysis method to identify the criticality of the different actions of an activity sequence under the assumption of stochastic action execution times. The results are returned to the designer in the form of a probability of criticality, providing a more informative view of system bottlenecks compared to the views produced by traditional methods.

The previous elements are joined in a design framework from which optimal logistical decisions for a batch of products of a flexible manufacturing system are computed from a modular specification. The logistics can be further improved by identifying and solving performance bottlenecks. The design framework is illustrated step-by-step using an academic case study, the Twilight system, and an industrial case study of a real ASML lithography scanner. The results show that the design flow fits the domain of flexible manufacturing systems and that optimal logistics decisions for good-weather behavior can be computed analytically. In cases where the timed state-space is not effectively computed it is shown that the use of over-specification can efficiently prune the timed state-space towards a computable size. In the industrial case-study the pruning reaches a 60% timed state-space reduction without loss of optimality. The work developed in this thesis marks a step forward in the model-based design of flexible manufacturing systems.

Contents

1 | Introduction

During the first and second industrial revolution, the manufacturing industry focused on developing systematic and efficient processes, using a mix of manual and automated labor, to achieve large-scale production of consumer goods. The coming of the digital revolution and the globalization effect drastically changed our society and the way we interact with the world. Products and technologies are being introduced and adopted at an ever-growing pace. This trend is observed in Figure 1.1 which depicts the penetration of different technologies and products in the United States of America from 1900 to 2005. In the figure we observe that newer technologies and devices are being adopted at faster rates throughout the decades and reaching very short adoption times for recent technologies. For instance, it took the smartphone ten years to reach market saturation (70% of the market share) [34], compared to twenty years for the mobile phone and around sixty years for land-line telephony. This trend led to an even more competitive market for manufacturing systems, where products require shorter time-to-market while also having shorter life-spans [53]. Modern manufacturers are further required to focus on customization and variety of products, besides performance and product quality [87].

By becoming more flexible and allowing custom and multiple product types to be manufactured within the same system, manufacturing systems are able to meet these market demands with a minimum impact on cost. This is achieved by adding system functionality, for example, in the form of multiple product routes within the system or by developing production units able to perform different operations on products [33]. These added functionalities make the systems more flexible to adapt to product changes. However, the incorporation of this flexibility leads to more involved system specifications. These specifications need to consider that system functions can be executed using multiple production units (resource sharing), possible product routes and production

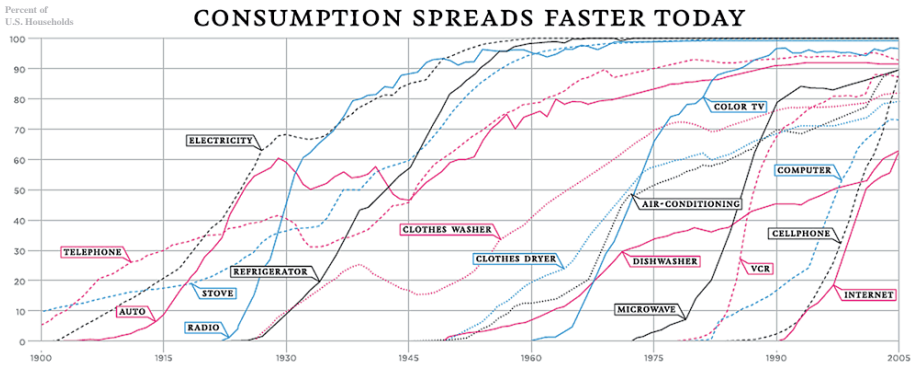


Figure 1.1: *Penetration of different technologies and products in the U.S. Market from 1900 to 2005, Source: Michael Felton, The New York Times [27].*

units assignments. Furthermore, these aspects also lead to a larger number of possible operation orderings and resource assignment choices. As a consequence, the optimization of operation orderings to manufacture a batch of products becomes more complicated. This thesis incorporates several contributions in the form of a design framework to address the challenges in the specification and design exploration of flexible manufacturing systems.

The remainder of this chapter is organized as follows. Section 1.1 introduces modern manufacturing systems. Section 1.2 discusses the added challenges of introducing flexibility in the specification and optimization of manufacturing systems. Section 1.3 discusses the design of manufacturing systems in its current state-of-practice and how a Model-based System Engineering approach could improve the speed at which design exploration converges towards a solution. Section 1.4 introduces the problem statement and enumerates the contributions of this work and Section 1.5 sketches the outline of the thesis.

1.1 Manufacturing Systems

A *manufacturing system* is an arrangement of machine and manual resources which transform materials, by the means of one or more physical processes, to manufacture a value-added product. This results in a final product (e.g. a car or a cellphone) or an intermediary part of another product (e.g. an integrated circuit of an electronic device or a door of a car). Modern manufacturing systems are driven by the economy of scale where the cost of the final product

is lowered due to mass production. These systems are designed and tuned to optimize the production of a single product type in a way that the cost of the system is amortized by each manufactured product. To ensure mass production, these systems rely on the use of machines and automation instead of on human and animal labor resources. The automation allows higher productivity and lower downtimes of production since machines can continuously operate for longer periods of time.

Manufacturing systems are usually composed of several *production* and *transportation resources*. Production resources perform *operations* on products, i.e. they realize the necessary physical processes to manufacture a product. For example, these operations could represent the printing of an integrated circuit on a silicon wafer or the assembly of different components of an integrated circuit. Transportation resources ensure the transfer of products between different production resources. This is commonly done with the use of conveyor belts and robot arms, and in some cases also using manual labor. Each product to be manufactured follows a specific recipe of operations. The necessary operations and the order in which they should be performed define the *product flow* of a product. For example, in semiconductor manufacturing a silicon wafer is first coated with a photoresistive material. The circuit pattern is then projected onto the coated wafer using a light source and lastly all non-exposed material is etched away to form the integrated circuit. Further, the product flow also dictates the possible routes that a product can have in the system as well as the resources that should perform the necessary operations.

The goal of modern manufacturing systems is to maximize productivity, ensure product quality and enable more product customization. These systems need to satisfy many *temporal* and *functional requirements* for quality and safety purposes. These requirements can emerge from the nature of physical processes. For example, making sure that the temperature of a product does not go below a certain bound or that a product is accurately oriented before an assembly operation. Requirements can also come from the system itself. For instance, ensuring that resources or product collisions do not occur or that resource capacities are respected.

Today's manufacturing systems can be viewed as a cyber-physical system which perform operations on batches of products. Production and transportation resources (the physical part) carry out operations while a logistics con-

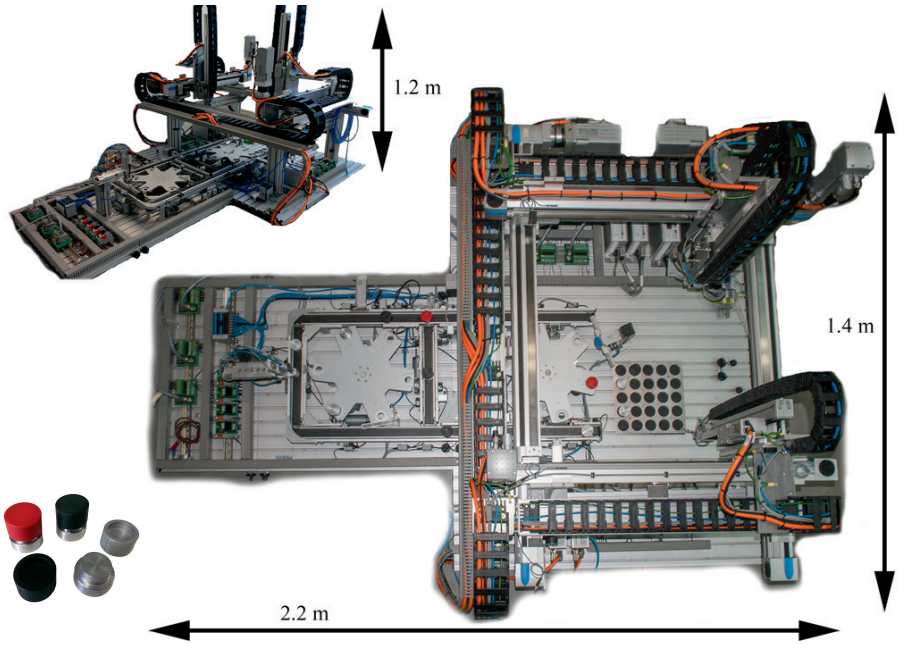


Figure 1.2: *xCPS manufacturing system and top and bottoms products [2].*

troller (the cyber part) assigns operations to resources and determines their order. The logistics controller of such a system instructs in which order operations are executed such that the makespan of a given batch is minimized and all requirements are satisfied. In a way, the physical part of the system can be seen as an orchestra, where each resource represents an instrument playing a specific role. In this context, the logistics controller acts as the conductor of the orchestra. Its mission is to coordinate the resources by indicating which resource needs to execute which operation and in what order it should happen. In a similar way, the conductor coordinates the instruments and musicians to play a complex piece of music in perfect harmony. This mission becomes more complicated when manufacturing systems are flexible, which is discussed in the following section.

1.2 Flexible Manufacturing Systems

With the coming of the digital revolution the global market demanded faster delivery times and more customization of products. Further, it needed greater

responsiveness to changes in products (such as corrections, improvements, maintenance) and production technologies (such as materials and physical processes). To adapt to the new era of manufacturing the concept of *flexible manufacturing systems* was introduced.

An example of a flexible manufacturing systems is the xCPS system (eXplore CPS) depicted in Figure 1.2. The xCPS is an assembly line emulator that is used for research and education in cyber-physical system domains [2], [4]. The system processes two types of cylindrical pieces, tops and bottoms, of different colors; the main use case of xCPS is to compose tops and bottoms into composite products, like the ones shown in the bottom left of Figure 1.2. The xCPS consists of one storage area, six conveyor belts, two indexing tables, two gantry arms, and several actuators and sensors. In its nominal production mode the system receives several individual bottom and top pieces and assembles them into a composite piece as a final product. Pieces are introduced in the system via the gantry arms which manage the storage area where bottoms and tops are stored as well as the finished composite pieces. Input pieces can have different orientations which need to be corrected during run-time. There are many possible other use cases of the xCPS platform due to its flexibility. Different conveyor belts can transport pieces through different paths including a re-entrance scenario where a piece can stay in the system without the use of the gantry arms. Different actuators are able to perform operations on pieces such as assembly, orientation correction, route changes and heating.

Flexible manufacturing systems, like the xCPS, gained much attention from researchers at the end of the 20th century. As a result, many different definitions emerged in literature. In [33] a literature review of different definitions of flexibility in manufacturing systems is done. Below we enumerate a selection of the notions of flexibility from [33] that we view as the most important features of the type of Flexible Manufacturing Systems (FMS) in this thesis:

1. *Machine flexibility*: the variety of operations offered by the system resources.
2. *Routing flexibility*: number of used routes / total number of possible routes across all resources.
3. *Process flexibility*: set of product types that can be produced without major set-up changes, i.e. product-mix flexibility.

Machine flexibility and *Routing flexibility* focus on the choice of resources and system layout. *Machine flexibility* considers the ability of system resources to perform different operations on products. For instance the gantry arm resources of the xCPS system are able to input and output pieces as well as organize them in the storage unit. *Routing flexibility* considers the number of possible routes across the resources in the system. For instance, in the xCPS case these would reflect the different routing possibilities enabled by the conveyor belts and route changing actuators. *Process flexibility* focuses on the flexibility of the product flow of the system. It considers the different number of product types that can be manufactured without major system changes (i.e. changing resources or the layout of the system). In the xCPS system this would correspond to the different types of recipes for the assembly of top and bottom pieces of different colors or different product flows. For example, composite pieces with a red top must pass by a heating station before output while composite pieces with a black top do not.

The different aspects of flexibility, in the layout, resources and in the product flow, have an impact on the design and optimization of the logistics controller of flexible manufacturing systems. The increase in flexibility leads to additional system functionalities that need to be taken into account in the functional system specification. These complex and large system specifications are difficult to write using classical monolithic specification approaches. Besides the functional aspects, flexible manufacturing systems also need to achieve demanding performance requirements. These are usually in terms of throughput (average product output per time unit) and makespan (the completion time of the manufacturing of a batch). The existence of multiple routes for products and resource assignments makes the problem of finding productivity optimal operation orderings for a batch of products more complicated. In this thesis we focus on the design of correct and makespan optimal logistic controllers. In the next section we discuss the specification and design exploration of flexible manufacturing systems and their logistics.

1.3 Specification and Design Exploration of Flexible Manufacturing Systems

During the early design stages of a manufacturing system different design alternatives are explored until all functional requirements are satisfied and the

expected performance is met (i.e. a minimal makespan or maximum throughput). The approach taken for this design exploration depends on the adopted design methodology. Traditional methodologies for system specification and design often rely on document-based approaches. When dealing with complex system specifications, the system is typically decomposed into multiple components (e.g. software components or mechanical components). For each component, multiple aspects must be considered such as behavior, timing, performance and accuracy aspects. In a document-based approach these components are often informally described without proper abstractions and lack separation of concerns (i.e. timing aspects and behavior aspects are coupled in the same specification). Further, the specification of the different aspects of a component are spread across multiple documents. These type of approaches often lead to inefficiencies in the design process. In this thesis we focus on addressing the inefficiencies with respect to specification and design exploration:

- **Specification**: typically documentation used for specification is in the form of text and diagrams, which do not describe the design intent in a formal, complete and consistent manner. Because of the lack of formality completeness and consistency checking cannot be properly supported, neither within the specification itself nor between the specification and the implementation of the system. As a consequence, in time specification and implementation typically start to run out sync until the specification has basically become useless. This severally hampers system evolvability.

- **Design Exploration**: The lack of formality of documentation makes it challenging to verify system requirements and predict performance properties and to assess the qualitative and quantitative impact of design decisions. As a consequence, the impact of incorrect specifications and non-optimal design decisions become apparent only during late stages of the design process, during which repair is time consuming and costly.

These disadvantages can be addressed by replacing documentation by models as the primary-citizens in the development process. This is the purpose of Model-based System Engineering (MBSE) approaches. System components, requirements and use-cases can all be specified as models depicting different aspects of the same system.

- **Specification**: MBSE models are formal. This implies that they are processable by computer algorithms and allow automated checks for consistency and completeness. In addition, they allow the automatic generation of artifacts such as implementation artifacts including code. In this way the consistency between specification and implementation is enforced by construction.

- **Design exploration**: MBSE models can also be the pivot from which mathematical models can be generated. These can be used to predict quantitative and qualitative properties of the system. In addition, such techniques allow the systematic exploration of design alternatives by either manually changing system specification or by automatically optimizing specification parameters.

There are many scientific results that studied different ingredients of MBSE approaches and their advantages. Without being exhaustive we enumerate some of these contributions. These include formalisms and techniques [3], [8], [28], [66], tools [18], [57], [64], [76], [78], methods and applications [9], [13], [44], [71], [73], [79], [81] and overviews [35], [43], [58], [67].

1.4 Problem Statement and Contributions

In the previous sections we set the stage by identifying the difficulties in the specification and design exploration of flexible manufacturing systems, as well as the shortcomings of traditional design approaches. The main problem statement driving the research and contributions of this thesis can be summarized as follows:

Developing a systematic approach for the specification and design exploration of complex Flexible Manufacturing Systems (FMSs) that is compatible with the industrial way of working.

In order to converge on a design specification for an FMS, it is important to be able to analyze and optimize its productivity. The productivity is highly dependent on aspects such as the geometrical layout, the routes of products between different resources, the performance of system operations and the mix of product types manufactured. Depending on the distances of resources and viability of routes between them, traveling times of products change and

possibly safety requirements are necessary to ensure that no collisions occur. Different product types being manufactured on the same system could imply the need to account for setup-times (i.e. amount of time required for a resource to adapt and perform a different operation). All these aspects play a role in the final performance of the system. Therefore any design exploration approach for FMSs should consider both the formal specification of these aspects, as well as allow for systematic design-space exploration and productivity optimization.

This thesis presents several contributions in the form of a design framework for the specification and design exploration of FMSs. The framework is developed with a MBSE methodology in mind to address the shortcomings of traditional design methodologies. The framework features a modular and compositional specification approach. Modularity enables the specification and analysis of the different modules in isolation, while compositionality permits us to reason about the properties of the whole system from the properties of its constituent modules. The specification approach uses formal models that are expressive enough to capture the characteristics of FMSs, such as the manufacturing of multiple product types, multiple routing options, and resource sharing. Further, it provides formal techniques to enforce functional requirements through synthesis and analyze the productivity of a given design specification. This enabling the early-stage design exploration of FMSs, by exploring and validating different system layouts and specifications.

The framework is developed such that it is consistent with the industrial way-of-working to facilitate its adoption by industrial practitioners. To this end, many concepts of the contributions of this thesis are inspired on the state-of-practice at industry. In particular, we rely on many concepts found in the design approaches, documentation and implementations of several wafer handling controllers of ASML. We either find solutions by developing new concepts or by combining and adapting existing state-of-the-art techniques in an effort to facilitate their integration in the current way-of-working found in industry. Due to this, many of the building blocks and architectural concepts of this thesis are inspired in the architecture and decomposition patterns used by the design approaches at ASML. This effort led to many of the results in this thesis to be integrated in a proprietary tool, developed by ESI (TNO) in collaboration with ASML, which is already in active use by ASML architects to design the latest lithography scanners. In the remainder of this section we introduce the different contributions and their relations in more detail.

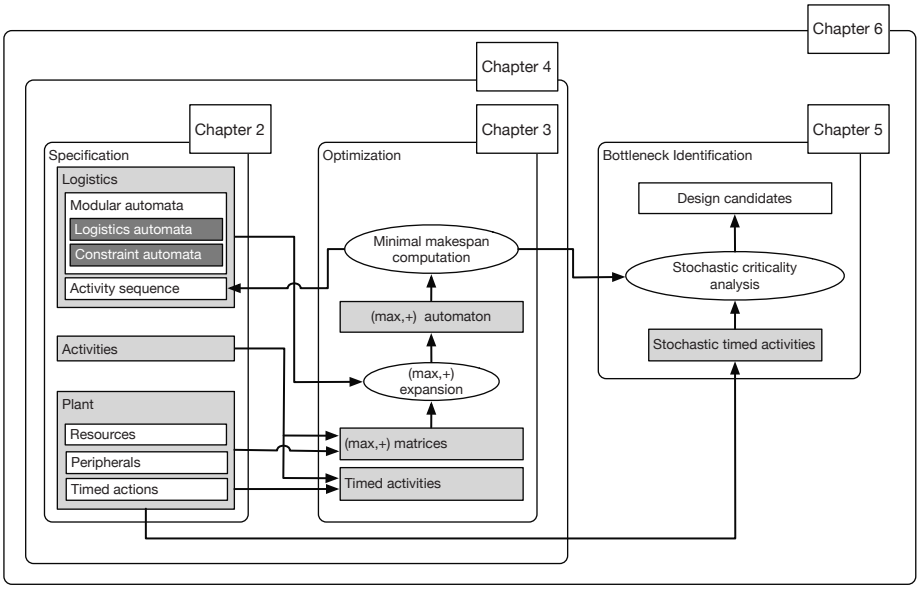


Figure 1.3: *Overview of the framework and thesis organization.*

Modular Specification of Flexible Manufacturing Systems (Ch. 2)

The first contribution of this thesis is a modular and compositional specification approach for FMSs and their logistics requirements. On the left of Figure 1.3, a box labeled *Specification* represents the abstraction and concepts for the specification of FMSs. The concepts are ordered hierarchically. The *plant* captures the decomposition of the system into a set of resources, peripherals, their actions and execution times. Resources represent transport or production units in an FMS, such as a robot arm. The peripherals represent the different physical components that compose the resource, such as motors and clamp peripherals. Finally, actions represent the interactions of different peripherals with the physical system, such as translations of motors and the clamping of products. On top of the plant a set of *activities* capture larger pieces of functional behavior of the system such as operations on products and transportation of products across the system. An activity is captured by a combination of peripheral actions and dependencies among those actions as a Directed Acyclic Graph (DAG). The complete manufacturing of a batch of products can then be captured by a specific ordering of these activities as an *activity sequence*, where each activity represents one operation of the FMS.

Logistics requirements concerning product flow are captured by finite state automata which we call a *logistics automata*. The language encoded by a logistics automaton represents the set of allowed activity sequences of the FMS for which a batch of products is correctly and safely manufactured. System requirements such as capacity constraints and safety constraints are capture by *constraint automata*. To cope with large and complex system specifications we allow for the modular specification of logistics and constraint automata by defining two operators: the composition operator and the constraining operator. The details of this modular specification of FMSs and their logistics requirements are explained in detail in Chapter 2. The abstractions and concepts chosen are inspired by the specification architecture and decomposition found in the state-of-practice at ASML. These specification concepts have been incorporated in a proprietary tool based on Domain Specific Languages (DSLs). This tool is already used by ASML architects in their development process. This contribution is based on the work published in [14], [15], [70] and [12].

Optimization of Flexible Manufacturing Systems (Ch. 3)

Given the specification of an FMS and its logistics requirements the next step is its productivity analysis and optimization. Therefore the second contribution of this thesis is an *optimization* approach to find the optimal makespan for a batch of products manufactured by a given FMS specification. In the middle of Figure 1.3 a box labeled *Optimization* depicts the concepts and methods used for the optimization domain of our framework such that different design specifications can be evaluated in terms of their expected makespan. We focus on flexible manufacturing systems which typically work with relatively small product batches and a mix of different product types. We introduce and define the Batch Makespan Optimization (BMO) problem and show it to fall within the class of NP-Hard problems. A solution to the BMO problem can be obtained by finding the activity sequence with the lowest makespan within the language of activity sequences of the specified logistics automaton. In order to efficiently compute the makespan of an activity sequence, we introduce (max,+) semantics for activities. The temporal behavior of each activity is captured by a single (max,+) matrix and an initial resource time-stamp vector. The makespan of an activity sequence is then efficiently computed by a series of (max,+) matrix multiplications. By the means of a *(max,+) expansion* algorithm we annotate the logistics automaton with the temporal characterization of activities to construct a *(max,+) automaton* for which each activity sequence represents the correct manufacturing of a batch of products. A solution for

the BMO problem can be obtained by finding an activity sequence in the state-space of the (max,+) automaton terminating in a final state with the lowest resource time-stamp vector norm. The details of this approach are given in Chapter 3. This approach has been used to validate the specification of an industrial wafer handling controller and is also integrated in the previously mentioned proprietary tool. This chapter is inspired by the throughput analysis approaches for Scenario-Aware Dataflow (SADF) models [39] and contributes with the adaptation of these analysis techniques to the domain of flexible manufacturing systems and with the complexity analysis of the BMO problem. These contributions are based on the work published in [70] and [14].

Exploiting Constraints to Reduce the Optimization-space (Ch. 4)

The framework is able to determine the optimal makespan activity sequence for a batch of products of a given system specification. The modularity of the framework allows for the specification of complex manufacturing systems and their requirements. However, the Batch Makespan Optimization (BMO) problem falls under the class of NP-Hard problems. As a consequence, finding optimal solutions might take prohibitively long depending on the size of the state-space of the corresponding (max,+) automaton. To cope with this complexity, our third and main scientific contribution of this thesis is an algebra of logistics automata to reason in a modular (algebraic) way about (behavioural and structural) *equivalence* and *inclusion* relations between logistics automata. This algebra of logistics automata allow us to systematically relate their languages, their state-space and (max,+) state-space sizes and their solutions to the BMO problem. Further, we introduce a systematic approach where we exploit the modularity of the framework by introducing additional requirements encoded as constraint automata in an effort to further reduce the state-space of (max,+) automata. This heuristic 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 additional constraints). The approach is inspired by common practices in an industrial setting, where manufacturing systems are typically over-specified [74] and in which over-specification is used implicitly and unconsciously to deal with complexity. Examples of over-specification that we have encountered in industrial cases are for instance: disallowing multiple mapping possibilities for an operation or enforcing the static ordering of system operations. These contributions are introduced in detail in in Chapter 4 and are based on the work publish in [14] and [17].

Bottleneck Identification using SCA (Ch. 5)

The goal of the framework presented in this thesis is the design exploration of flexible manufacturing systems. This includes exploring different types of resources and peripherals and different system layouts. This includes geometrical locations of resources, the number of resources, and the routing options concerning the product flow. The fourth contribution of this thesis is an approach to identify performance bottlenecks in the logistics of a flexible manufacturing systems, even when peripheral actions may exhibit stochastic execution times. On the right of Figure 1.3 a box labeled *Bottleneck identification* depicts the concepts and methods used for the proposed bottleneck identification analysis. We start by extending our framework with stochastically timed actions and activities. Further, we introduce Stochastic Criticality Analysis to estimate the criticality of the actions of an activity by identifying how often certain actions appear on the critical path. The likelihood of an action appearing on the critical path is directly related to how likely this action is to be a performance bottleneck of the system. This approach takes inspiration from the concept of Criticality Index introduced in the field of project and planning [33]. The contribution of this chapter is the formalization of the Stochastic Criticality Analysis (SCA) approach with formal mathematical support together with confidence intervals to obtain results with known accuracy and its application to the domain of FMSs. This approach has also been integrated in the previously mentioned proprietary tool. The details of these contributions are discussed in depth in Chapter 5 and are based on the work published in [16].

Industrial Case Study (Ch. 6)

The final contribution of this thesis is an application of this framework that demonstrates its scalability to an industrial logistics controller. We will apply our framework to specify the system, optimize its makespan and identify performance bottlenecks. For this purpose, we start by a specification of the plant, activities and logistic requirements of the system using the concepts introduced in Chapter 2. We then find the optimal makespan activity sequence for a batch of wafers using the technique of Chapter 3 and further reduce the optimization-space by more than 60% by exploiting over-specification and following the method introduced in Chapter 4. Finally we select candidates for design improvement by identifying performance bottlenecks in the makespan optimal activity sequence using the Stochastic Criticality Analysis of Chapter 5. The details of this case-study are presented in Chapter 6. Similar case studies are being performed ‘as we speak’ by ASML architects to design and optimize

the logistics of ASML lithography scanners using the previously mentioned proprietary tool integrating many of the results of this thesis.

1.5 Thesis Overview

The remainder of this thesis is organized as follows. Chapter 2 introduces the modular specification approach for FMSs and their logistics. Chapter 3 discusses the optimization of a given specification by introducing the Batch Makespan Optimization (BMO) problem and an approach to solve it. Chapter 4 discuses the growth of the optimization-space of a given BMO and defines an algebra of logistics automata to reason about the behavior and state-space sizes of logistics automata and a systematic approach for the specification of logistics that ensures the reduction of the optimization-space. Chapter 5 introduces stochastic criticality analysis as a means to identify performance bottlenecks in FMSs assuming that actions may exhibit stochastic execution times. Chapter 6 uses the concepts of Chapter 2, 3, 4 and 5 to specify, optimize and identify bottlenecks in an industrial case study. Finally Chapter 7 discusses the main conclusion and future work.

2 | Modular Specification of Flexible Manufacturing Systems

In the previous chapter we sketched the overall design flow and domains of the contributions of this thesis. In this chapter we dive into the *specification* domain and show how our framework captures the components and functionalities of modern flexible manufacturing systems. Figure 2.1 depicts the concepts used in the specification domain. These are organized in three groups, *plant*, *activities* and *logistics*. The plant abstracts the physical components of the system in terms of *resources*, *peripherals* and *actions*. On top of the plant, *activities* can be constructed to define certain functional behaviors of the system, such as transport or manufacturing operations. An *activity sequence* describes 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. An activity captures how an operation is executed and which resources are used while an activity sequence captures in which order the activities are executed, i.e. the product flow. The set of all possible activity sequences is captured as the language accepted by a finite state automaton, which we call a *logistics automaton*. Logistics automata are used to capture desired requirements on the ordering and choice of activities, concerning for example product flow and assignment to resources. Logistics automata can be defined modularly for each product in a batch. The resulting logistics automaton of a batch can be obtained via the composition of the individual logistics requirements for each product using a *composition operator* on logistics automata. 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. This modularity enables our framework to tackle the complex description of modern

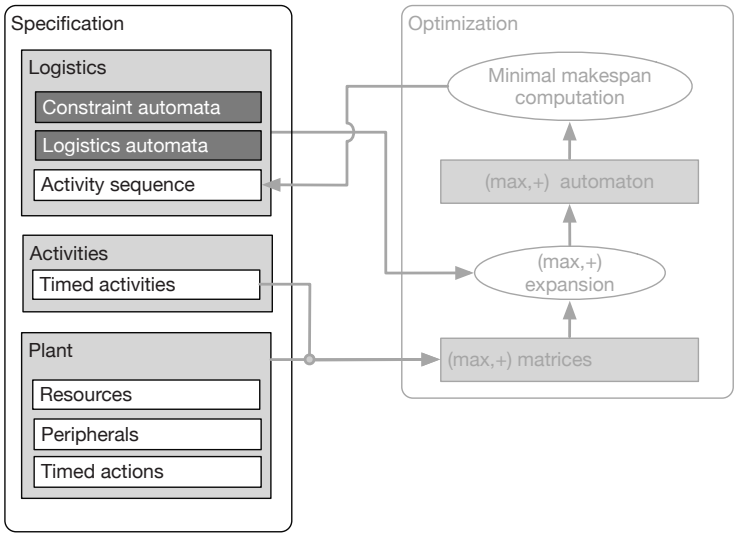


Figure 2.1: *Overview of the framework concepts for the* Specification *domain.*

manufacturing systems. Finally, the *specification* concepts introduced here are also used as the foundation for the optimization steps discussed in Chapters 3 and 4, as well as the design-exploration technique presented in Chapter 5. This chapter is organized as follows. In Section 2.1 we introduce the Twilight system as a running example to showcase the application of the specification concepts. Then we will address the different specification concepts following their hierarchy as depicted in Figure 2.1, the plant model in Section 2.2, the activities in Section 2.3 and the logistics in Section 2.4. Section 2.5 discusses the relevant related work and finally Section 2.6 concludes.

2.1 The Twilight System

Examples of a complex modern manufacturing system are the lithography systems built by ASML[1]. The Twilight system is a simplification of a wafer handling sub-system used at ASML which we discuss in detail in Chapter 6. The constructed Twilight system reflects most of the challenges of the design exercise but removes unnecessary domain-specific content and implementation details. It uses similar kinds of peripherals and resources and a similar product flow but on a smaller scale. Therefore, it is suitable to use as running example

[1]Supplier of photo-lithography systems, www.asml.com

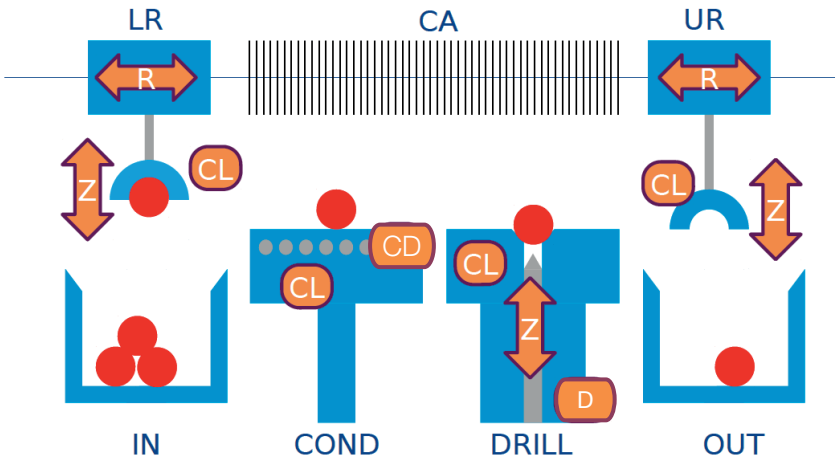


Figure 2.2: *Twilight manufacturing system and its resources and peripherals [70].*

to illustrate the concepts and techniques used throughout this thesis.

R *The name Twilight System originates from the family name of the student who created it. This name,* 'Duisters' *roughly translates from Dutch to English as 'Twilight' [31].*

2.1.1 System Purpose

The Twilight system is depicted in Figure 2.2. The products in the system are *balls*, shown as red circles in the figure. The manufacturing purpose is to engrave a certain profile on each of the balls of a batch using a *drill* device. This profile can vary in terms of depth or pattern and these are defined by a given user recipe. Complex engraving profiles might require multiple drilling operations. Therefore, a batch could possibly re-enter the system multiple times. For this purpose, before the drilling operation, each ball is subjected to certain *conditioning* operations. These align the balls for correct overlay of multiple profiles and set the temperature of the balls for ideal drilling conditions. Besides ensuring that products are manufactured correctly, the system also needs to meet performance goals. For the twilight this is to achieve the lowest possible makespan for any given batch of products.

2.1.2 Resources and Peripherals

The system contains seven resources depicted in Figure 2.2 as IN, OUT, LR, UR, CA DRILL and COND. There are two buffer resources for input (IN) and output

(OUT) of products. There are two robots which are used to transport the ball products between the different processing units and input/output buffers, the Load Robot (LR) and the Unload Robot (UR). Since the LR and UR share some common areas (above the COND and DRILL resources) we consider this shared area as a resource which we call Collision Area (CA). The remaining resources are two processing units, the Conditioner (COND) and the Drill (DRILL). The Conditioner ensures that each ball is conditioned to a predefined temperature and is correctly aligned for the drilling operation. The Drill unit is responsible for the engraving of the profiles on a ball.

The peripherals of each resource are depicted as orange symbols drawn in the figure. Both robots are composed of three peripherals; a clamp (CL) to pick up and hold a ball, an R-motor (R) to move along the rail, and a Z-motor (Z) to move the clamp up and down. The conditioner is composed by a clamp (CL) and a conditioner (CD) peripheral, to respectively heat and align the ball. The drill is composed by a drill bit (D) that performs the drill action, a clamp (CL) and a Z-motor (Z) to move the drill bit up and down.

2.1.3 Product Flow

Each batch follows the same product flow for every ball in the batch. First, a ball is picked up at the input buffer. Then it is brought to the conditioner and processed. Next, the item is transported to the drill, where the profile is engraved. Finally, the drilled ball is transported to the output buffer. Once the last ball has been processed the batch is either finalized or it re-enters the system in case it requires additional engraving.

Every operation of the product flow is assigned to a specific resource. Some operations have a unique assignment. The conditioning and drilling operations can only be performed on the COND and DRILL resources respectively. Moreover, due to the range of the robots, both the input and output operations are assigned respectively to the LR and UR resources. However, the transport of products between the COND and DRILL resources can be executed by either the LR or the UR.

2.1.4 Locations and Collision Area

The transportation of products among the different processing units is done by the LR and UR resources. For this purpose, certain locations are predefined to describe the positions and movement paths of the two robots. All reachable

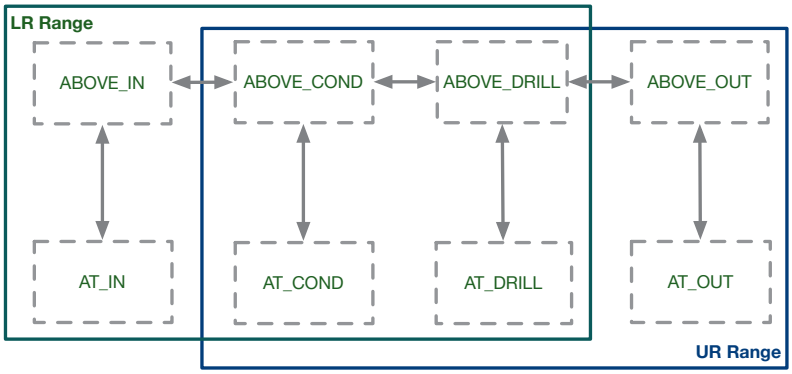


Figure 2.3: *Locations and robot movement ranges in the Twilight system (in accordance with the resource layout of Figure 2.2).*

locations and paths of each robot are shown in Figure 2.3. Note that for resources IN, COND, DRILL and OUT, there are two vertical positions, ABOVE and AT. The attribute AT refers to locations where a robot picks/places a product on a processing unit. Once the robot is done with the product exchange it retracts to the corresponding ABOVE location. Both robots are able to reach resources COND and DRILL. Therefore, the overlapping reachable locations of both robots could lead to collisions. In order to avoid this, we consider this shared area as a resource (CA) which robots must claim and release before accessing it. This area is depicted between the LR and UR in Figure 2.2. Finally, each robot has a predefined homing position: LR on the left corner (ABOVE_IN) and UR on the right corner (ABOVE_OUT). By default, robot movements always start and end at the homing position.

2.1.5 Requirements

On top of the product flow there are certain functional requirements on the batch manufacturing of the Twilight system due to resource and safety constraints. These are enumerated bellow:

- Products shall enter and leave the system in a First-In-First-Out (FIFO) order.
- There shall only be one product at a time in resources LR, UR, COND and DRILL (unary capacity).
- Products shall not collide (i.e. products shall not be be placed on a occupied resource).

- Resources shall not collide (i.e. the LR and UR shall never share the same location).

In the remainder of this chapter, we introduce the specification concepts of Figure 2.1 and show how we can model these manufacturing characteristics using the Twilight system as an application example.

2.2 Plant

In our framework (see Figure 2.1) we start the system specification by defining the *plant*. The plant defines the basic structure and elementary actions of the system. It serves as the lowest abstraction layer of our framework and defines the offered actions of the system. *Activities* are then built based on the plant elements.

A plant consists of several *resources*, such as a robot arm or a processing unit. A resource is composed of zero or more *peripherals*. Each peripheral can execute *actions* where an *action* describes an atomic behavior of the system, e.g. the movement of a motor or the activation or deactivation of a clamp. Each action is associated with an execution time. The complete set of actions captures all behavior that the system can exhibit. The basic elements of our plant specification are formally defined by the following sets:

- set $\mathcal{A}$ of actions, with typical elements $x, y, z \in \mathcal{A}$;
- set $\mathcal{P}$ of peripherals, with typical elements $p \in \mathcal{P}$;
- set $\mathcal{R}$ of resources, with typical elements $r \in \mathcal{R}$.

We further assume a function $R : \mathcal{P} \to \mathcal{R}$, to reflect that $R(p)$ is the resource that contains peripheral p, and a function $T : \mathcal{A} \to \mathbb{R}_{\geq 0}$ that maps each action to its fixed execution time.

Example 2.1 Consider the Load Robot (LR) resource of the Twilight system shown in Figure 2.2 which can move products within the system. This robot is composed from three peripherals. These are identified by the orange symbols drawn over and around the resource: the R and Z motors and the CL clamp. The R and Z motor allow the LR to move within the defined locations. The first accounts for horizontal movements and the later

for vertical movements. The CL clamp represents an on/off actuator that is able to grasp/release a ball product. Actions of these peripherals can be a single motor translation to a location, or the on/off actions of the clamp peripheral.

2.2.1 Application to the Twilight

Let us illustrate the plant specification using the Twilight system. Table 2.1 lists all the actions provided by the system. To facilitate their referencing each action is given a short alias shown within parentheses before each action name in Table 2.1. Some of the provided actions can only be associated with specific peripherals (e.g. *condition* action with peripheral CD of the COND resource) while other actions can be associated with multiple peripherals (e.g. *clamp_on* action with peripherals CL of resources LR, UR, DRILL and COND). Actions associated with the movement of robots (e.g. LR_mv_COND_to_DRILL and UR_mv_DRILL_to_COND) are duplicated in this specification for common locations of the LR and UR resources because we assume that peripherals of these robots can have different movement speeds and thus different execution times. Note that these actions are defined by a single movement from one symbolic location to another adjacent location (adjacent locations are connected by direct arrows between locations in Figure 2.3). This decision is made based on the geometrical layout of Figure 2.3 where reachable locations are predefined. A different geometrical layout would imply a redefinition of the affected system actions. For example, moving from location AT_COND to location ABOVE_COND implies the execution of action l5 if using the LR (or u5 in case of using the UR). A more elaborate movement such as moving from location ABOVE_COND to AT_DRILL implies the sequential execution of actions l3 and l6 if using the LR (or u1 and u6 in case of using the UR).

Table 2.2 depicts the decomposition of the Twilight system into resources and peripherals. The seven resources are listed along with the associated peripherals as depicted in Figure 2.2 by the orange labels on and around each resource. For instance, the COND resource is composed of two peripherals: a conditioner peripheral COND.CD and a clamp peripheral COND.CL. Notice that the CA, IN and OUT resources do not have any peripherals and thus are not able to perform any actions.

Table 2.1: *Twilight system actions.*

(l1) LR_mv_COND_to_IN	(u1) UR_mv_COND_to_DRILL
(l2) LR_mv_IN_to_COND	(u2) UR_mv_DRILL_to_COND
(l3) LR_mv_COND_to_DRILL	(u3) UR_mv_OUT_to_DRILL
(l4) LR_mv_DRILL_to_COND	(u4) UR_mv_DRILL_to_OUT
(l5) LR_mv_AT_to_ABOVE	(u5) UR_mv_AT_to_ABOVE
(l6) LR_mv_ABOVE_to_AT	(u6) UR_mv_ABOVE_to_AT
(d1) move_UP	(d2) move_DOWN
(d3) drill_on	(c1) condition
(cl1) clamp_on	(cl2) clamp_off

Table 2.2: *Twilight resources and peripherals.*

Resource	**Peripherals**
(LR) Load Robot	LR.Z, LR.R and LR.CL
(UR) Unload Robot	UR.Z, UR.R and UR.CL
(DRILL) Drill	DRILL.Z, DRILL.D and DRILL.CL
(COND) Conditioner	COND.CD and COND.CL
(CA) Collision Area	-
(IN) Input buffer	-
(OUT) Output buffer	-

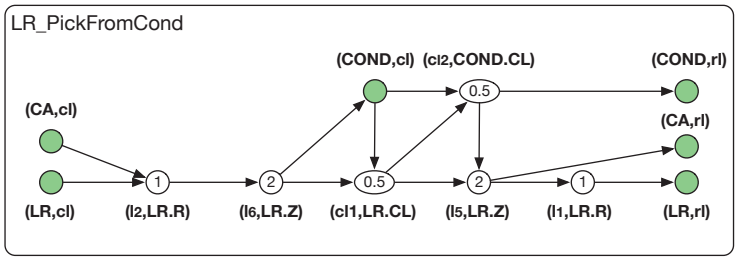


Figure 2.4: *Activity LR_PickFromCond: Resource COND and LR are used to place a product from the LR to the COND unit.*

2.3 Activities

On top of the plant specification we can describe more elaborate functional behaviors of the system by combining multiple actions and defining dependencies among those actions. For instance, to describe a movement from AT_IN to ABOVE_DRILL by the Load Robot resource, we define a sequential execution of actions l5, l2, l3 and l6. In our framework, such a combination of actions is called an *activity*.

Definition 2.1 — (Activity). An activity is a directed acyclic graph $(N, \rightarrow)$, consisting of a set N of nodes and a set $\rightarrow \subseteq N \times N$ of dependencies. We write a dependency $(n_1, n_2) \in \rightarrow$ as $n_1 \rightarrow n_2$. We assume a mapping function $M : N \rightarrow \mathcal{A} \times \mathcal{P} \cup \mathcal{R} \times \{rl, cl\}$, which associates a node to either a pair (x, p) referring to an action x executed on a peripheral p; or to a pair (r, v) with $v \in \{rl, cl\}$, referring to a *claim* (cl) or *release* (rl) of resource r. Nodes mapped to a pair (x, p) are called *action nodes*, and nodes mapped to a claim or release of a resource are called *claim* and *release* nodes respectively.

For every action in an activity the resource on which its execution takes place must be claimed before the action is executed and released once the activity is finalized. This ensures that while an activity is executing the resources cannot be claimed by another activity. The set of resources used by an activity is defined as follows:

Definition 2.2 — (Resources of Activity). Given activity $a = (N, \rightarrow)$, we define the set $R(a) = \{r \in \mathcal{R} \mid (\exists n \in N \mid M(n) = (r, cl))\}$.

Example 2.2 Consider the activity LR_PickFromCond depicted in Figure 2.4. Nodes are represented with circles and release and claim nodes are colored in light green. Dependencies are depicted as directed edges between nodes. The activity requires resources LR, COND and CA. Every action node is labeled with a tuple indicating the action and peripheral. For example, label (l2,LR.R) indicates that this node refers to action l2 which is executed on peripheral LR.R. Finally, the execution time of each action is shown within the corresponding node (with the exception of the claim and release nodes for which we assume an execution time of 0 time units). The function of activity LR_PickFromCond is that the LR picks a product from the COND processing unit. Since this operation requires access to the shared area the CA resource is claimed before the first robot movement and released after the last movement. The activity starts by claiming the LR resource and the CA resource. Using peripheral LR.R, the activity movies the LR resource from its home position ABOVE_IN to the ABOVE_COND position (action node (l2,LR.R)) and then immediately to location AT_COND (action node (l6,LR.Z)). Then the product handover takes place. After claiming resource COND the clamp of the receiving resource LR clamps the product and after the clamp of the delivering resource COND unclamps it (action nodes (cl1,LR.CL) and (cl2,COND.CL) respectively). Finally the LR returns to its homing location using peripherals LR.R and LR.Z (action nodes (l5,LR.Z) and (l1,LR.R) respectively).

We define several conditions that activities must satisfy to statically check for proper resource claiming and releasing within an activity as well as proper ordering of actions executed on the same peripheral. Let $(N, \rightarrow)$ be an activity and let $\rightarrow^+$ be the transitive closure of $\rightarrow$, then:

1. All nodes mapped to the same peripheral are sequentially ordered:
   $\forall n_1, n_2 \in N, x_1, x_2 \in \mathcal{A}, p \in \mathcal{P} : n_1 \neq n_2 \implies$
   $(m(n_1) = (x_1, p) \wedge m(n_2) = (x_2, p)) \implies (n_1 \rightarrow^+ n_2 \vee n_2 \rightarrow^+ n_1)$
2. Each resource is claimed no more than once:
   $\forall n_1, n_2 \in Nr \in \mathcal{R} : (m(n_1) = (r, cl) \wedge m(n_2) = (r, cl)) \implies n_1 = n_2$
3. Each resource is released no more than once:
   $\forall n_1, n_2 \in N, r \in \mathcal{R} : (m(n_1) = (r, rl) \wedge m(n_2) = (r, rl)) \implies n_1 = n_2$

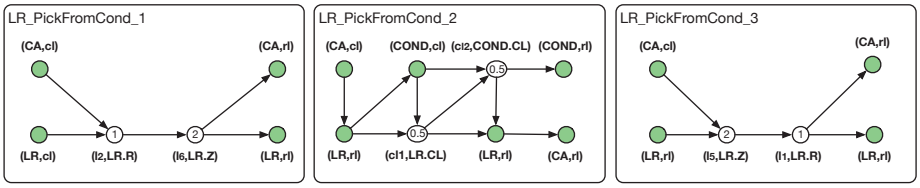


Figure 2.5: *Set of three activities which executed sequentially have equivalent behavior as activity LR_PickFromCond depicted in Figure 2.4.*

4. Every action node is preceded by a claim node on the corresponding resource:
$$\forall n_1 \in N, x \in \mathcal{A}, p \in \mathcal{P}, r \in \mathcal{R} : (m(n_1) = (x, p) \wedge\ R(p) = r) \implies (\exists n_2 \in N : m(n_2) = (r, cl) \wedge n_2 \rightarrow^+ n_1)$$
5. Every action node is succeeded by a release node on the corresponding resource:
$$\forall n_1 \in N, x \in \mathcal{A}, p \in \mathcal{P}, r \in \mathcal{R} : (m(n_1) = (x, p) \wedge\ R(p) = r) \implies (\exists n_2 \in N : m(n_2) = (r, rl) \wedge n_1 \rightarrow^+ n_2)$$
6. Every release node is preceded by a claim node on the corresponding resource:
$$\forall n_2 \in N, x \in \mathcal{A}, r \in \mathcal{R} : m(n_2) = (r, rl) \implies (\exists n_1 \in N : m(n_1) = (r, cl) \wedge n_1 \rightarrow^+ n_2)$$
7. Every claim node is succeeded by a release node on the corresponding resource:
$$\forall n_1 \in N, x \in \mathcal{A}, r \in \mathcal{R} : m(n_1) = (r, cl) \implies (\exists n_2 \in N : m(n_2) = (r, rl) \wedge n_1 \rightarrow^+ n_2)$$

2.3.1 On Designing Activities

Even though an activity represents a piece of deterministic functional behavior, the amount of behavior to be included in this activity can be determined in different ways. As an extreme case, an activity could be defined as a single action (together with the corresponding resource claim and release). On the other hand, a single activity could encompass all system behavior needed to produce a batch of products. Determining the granularity of activities depends on system layout, resource sharing and also on the level of performance optimization desired.

Example 2.3 Recall the activity defined for the Twilight system depicted in Figure 2.4. The behavior of this activity can be capture as a set of multiple smaller activities as exemplified by the set of three activities depicted in Figure 2.5. LR_PickFromCond_1 describes the movement from the homing location ABOVE_IN to ABOVE_COND, LR_PickFromCond_2 the handover of the product and a LR_PickFromCond_3 describes the return of LR to its homing location ABOVE_IN. If executed in sequence, these activities exhibit the same behavior as activity LR_PickFromCond of Figure 2.4. The semantics of executing a sequence of such activities is formalized in Section 2.4.1.

2.3.2 Timed Activities

Besides capturing pieces of end-to-end deterministic functional behavior of the system, activities also encode execution timing information. This is done assuming an As-Soon-As-Possible (ASAP) execution of actions, respecting dependencies and the claiming and releasing of resources. To achieve this, we rely on the notions of *synchronization* (an action can only start once all its dependencies are satisfied) and *delay* (the execution of an action takes a fixed amount of time). In order to define the temporal execution of activities we start with the notion of the execution time of an action and a node, which we use to formalize the notion of *delay*.

Definition 2.3 — (Execution time of a node). Given an Activity $a = (N, \rightarrow)$ we define a function $T : N \rightarrow \mathbb{R}_{\geq 0}$ that maps each node to a fixed execution time. Given a node $n \in N$:

$$T(n) = \begin{cases} T(x) & \text{if } M(n) = (x, p) \\ & \text{for some } x \in \mathcal{A}, p \in \mathcal{P} \\ 0 & \text{otherwise.} \end{cases}$$

We further define the notion of predecessor nodes, which we use to formalize the notion of *synchronization*.

Definition 2.4 — (Predecessor nodes). Given activity $(N, \rightarrow)$ and node $n \in N$, we define the set of predecessor nodes of n as:

$$Pred(n) = \{n_{in} \in N \mid n_{in} \rightarrow n\}.$$

Since actions are executed on peripherals belonging to a specific resource, we assume a *resource time-stamp* function $\gamma_R : \mathcal{R} \to \mathbb{R}^{-\infty}$, where $\mathbb{R}^{-\infty} = \mathbb{R} \cup \{-\infty\}$. The function represents the system configuration in terms of resource availability, mapping to each resource $r \in \mathcal{R}$ an entry $\gamma_R(r) \in \mathbb{R}^{-\infty}$ corresponding to the availability time of resource r. These entries are used to determine when resources are available, and hence can be claimed. All entries in the initial configuration of the system are assumed to be zero, to indicate that all resources are available upon system start. We denote the initial configuration as $\mathbf{0}_R$, implying that $\forall r \in \mathcal{R} : \mathbf{0}_R(r) = 0$. We now define the *start* and *end time* of a node, given an initial resource time-stamp function.

Definition 2.5 — (Start and end time of a node). Given activity $a = (N, \to)$ and resource time-stamp function γ_R, we define the start time $start(n)$ and end time $end(n)$ for each node $n \in N$:

$$start(n) = \begin{cases} \gamma_R(r) & \text{if } M(n) = (r, cl) \\ \max\limits_{n_{in} \in Pred(n)} end(n_{in}) & \text{otherwise} \end{cases}$$
$$end(n) = start(n) + T(n).$$

A node n starts as soon as all predecessor nodes have completed execution. Note that the start and end times for each node are uniquely defined, due to the structural condition of activities defined in Section 2.3. This also means that the execution semantics of an activity $a = (N, \to)$ are uniquely defined by N, $\to$, timing function T, and resource time-stamp function γ_R.

This unique temporal execution of an activity can be represented as a schedule using a Gantt chart, for which the horizontal axis represents the actions executed in time and the vertical axis shows the horizontal lanes for each peripheral. Each box labeled x in an horizontal lane p represents the execution of a node n mapped to peripheral p and action x (i.e. $m(n) = (x, p)$). The length of the box is equivalent to $T(n)$ starting at time $start(n)$ and finishing at time $end(n)$.

Now, consider a resource time-stamp function γ_R as the starting configuration of the availability of system resources. The execution of activity $a = (N, \to)$, assuming the starting configuration γ_R, will lead to a new configuration of

the availability of system resources. We define the *update* of the resource time-stamp function due to the execution of an activity a as follows:

Definition 2.6 Given activity $a = (N, \rightarrow)$ and resource time-stamp function γ_R we define the update $update(\gamma_R, a)$ of the resource time-stamp function due to the execution of activity a as:

$$update(\gamma_R, a)(r) = \begin{cases} \gamma_R(r) & \text{if } r \notin R(a) \\ end(n) & \text{if } r \in R(a) \wedge M(n) = (r, rl) \\ & \quad \text{for some } n \in N. \end{cases}$$

The updated resource time-stamp function represents the availability times of all system resources after the execution of the activity.

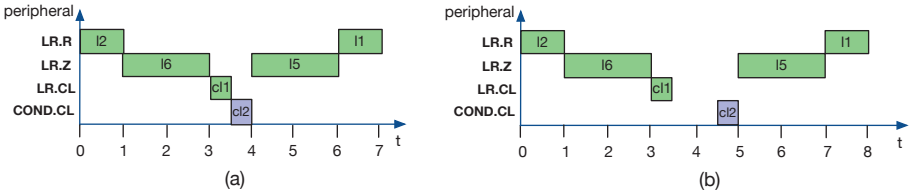


Figure 2.6: *Gantt chart of the temporal execution of activity LR_PickFromCond depicted in Figure 2.4 for two different resource time-stamps (a) $\mathbf{0}_R$ and (b) $\gamma_R(LR) = 0$ and $\gamma_R(COND) = 4.5$.*

Example 2.4 Consider the activity LR_PickFromCond depicted in Figure 2.4. Two different temporal executions are shown in the Gantt charts of Figure 2.6 (a) and (b). We assume an initial configuration $\mathbf{0}_R$ for Figure 2.6 (a) and configuration γ_R with $\gamma_R(\text{LR}) = 0$, $\gamma_R(\text{COND}) = 4.5$ and $\gamma_R(\text{UR}) = \gamma'_R(\text{IN}) = \gamma_R(\text{OUT}) = \gamma_R(\text{DRILL}) = \gamma_R(\text{CA}) = 0$ for Figure 2.6 (b). The result of $update(\mathbf{0}_R, a)$ gives us a new function γ'_R for which $\gamma'_R(\text{LR}) = 7$ and $\gamma'_R(\text{COND}) = 4$. In the case of (b) the different configuration leads to a delay of action node (cl2,COND.CL). This delay induces a delayed activity completion time since action node (l5,LR.Z) needs to synchronize and thus is also delayed. In this case $update(\gamma_R, a)$ gives us a new function γ'_R for which $\gamma'_R(\text{LR}) = 8$ and $\gamma'_R(\text{COND}) = 5$. In both cases the availability times

of the unclaimed resources of the activity after execution remain the same, i.e. $\gamma'_R(UR) = \gamma'_R(IN) = \gamma'_R(\text{OUT}) = \gamma'_R(\text{DRILL}) = \gamma'_R(CA) = 0$.

The makespan of an activity is captured by the total time elapsed since the start time of the first action to the end time of the last action. This is determined by the largest value in the updated resource time-stamp function after executing the activity with initial configuration $\mathbf{0}_R$. We define the *makespan* of an activity as follows:

Definition 2.7 — (Makespan of an activity). Given an Activity a and initial resource time-stamp function $\mathbf{0}_R$ we define the makespan of a as:

$$mks(a) = \left\| update(\mathbf{0}_R, a) \right\|$$

where for any resource time-stamp function γ_R, $\|\gamma_R\|$ is the norm of γ_R defined by $\|\gamma_R\| = \max_{r \in \mathcal{R}} \gamma_R(r)$.

Example 2.5 Consider again activity LR_PickFromCond depicted in Figure 2.4 for which its temporal execution assuming $\mathbf{0}_R$ is shown in Figure 2.6 (a). The makespan of activity LR_PickFromCond is equal to $mks(\text{LR\_PickFromCond}) = 7$ since the largest value of the resource time-stamp function after the update function is 7. This is also visible in the Gantt chart since the the last action node to complete its execution is (l1,LR.R).

2.3.3 Application to the Twilight

Table 2.3 lists all the activities we defined for the Twilight System (including activities LR_PickFromCond and UR_PutOnCond depicted in Figures 2.4 and 2.7). Each activity is associated with a numerical alias shown in bold before the activity name to facilitate their referencing. The functional granularity of

Table 2.3: *Set of activities of the Twilight System.*

(**1**) LR_PickFromInput	(**5**) LR_PutOnDrill	(**9**) UR_PutOnCond
(**2**) LR_PutOnCond	(**6**) UR_PickFromDrill	(**10**) UR_PutOnOutput
(**3**) LR_PickFromCond	(**7**) UR_PickFromCond	(**11**) Condition
(**4**) LR_PickFromDrill	(**8**) UR_PutOnDrill	(**12**) Drill

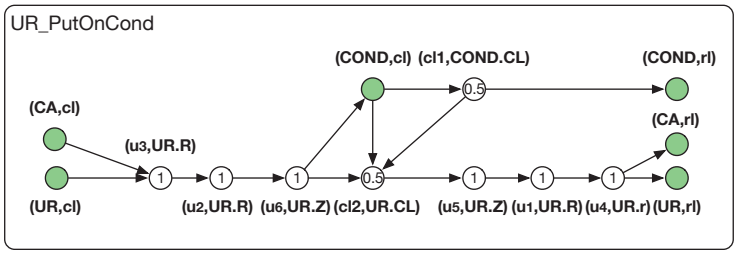


Figure 2.7: *Twilight activity UR_PutOnCond.*

the activities defined is based on the actions to be performed on the product and desired product flow (Section 2.1.3). For the robots (LR and UR) activities are based on the placing and picking of products from the different resources and input and output buffers (Activities **1** to **10**). Additionally two activities are defined for the two operations of the COND and DRILL processing units (Activities **11** and **12**). Given the set of activities one can specify the complete manufacturing of a product as a sequence of activities, for example sequence **1 2 11 3 5 12 6 10**. In the next section we introduce and further explain the notions of activity sequence and sequencing operator. The complete specification of the activities of the Twilight system can be found in Appendix A.

2.4 Logistics

An activity sequence can model the complete manufacturing *logistics* of a batch of products. Multiple sequences of activities can encode the correct and complete manufacturing of a batch of products. Hence the logistics is defined as a collection of allowed *activity sequences*. This collection can be encoded as the language accepted by a finite state automaton, which we call *logistics automaton*. In order to capture constraints across different product flows, such as resource capacity and input order constraints, we introduce *constraint automata*. Logistics automata and constraint automata are composed using specific operators permitting a modular specification of manufacturing logistics of batch-oriented manufacturing systems. In this section we introduce the concepts of *activity sequence*, *logistic automata*, *constraint automata* and the corresponding *composition* and *constraining* operators, and show how these concepts are used to modularly specify the manufacturing logistics of flexible manufacturing systems.

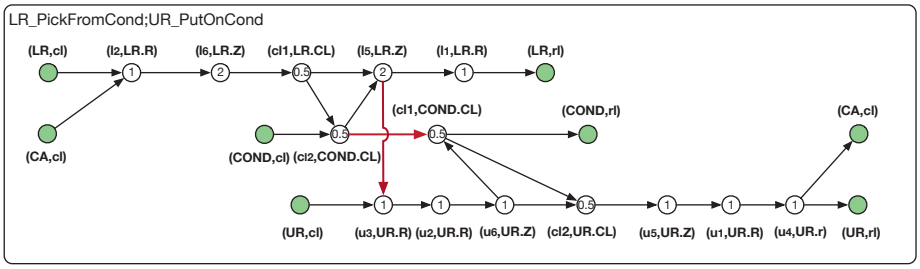


Figure 2.8: *Resulting activity LR_PickFromCond;UR_PutOnCond from the sequencing of LR_PickFromCond and UR_PutOnCond.*

2.4.1 Activity Sequences

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 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 defining a *sequencing operator* that the combines two activities in a new activity.

Definition 2.8 — (Sequencing Operator). Given two activities $a_1 = (N_1, \rightarrow_1)$ and $a_2 = (N_2, \rightarrow_2)$ with $N_1 \cap N_2 = \emptyset$, we define $a_1; a_2$ as activity $a_{1;2} = (N_{1;2}, \rightarrow_{1;2})$. Before we define $N_{1;2}$ and $\rightarrow_{1;2}$, we define $R_{1\cap 2} = R(a_1) \cap R(a_2)$ as the set of resources used in both activities. Further we define the set of corresponding release nodes in N_1, and claim nodes in N_2 as: $rl_{1\cap 2} = \{n_1 \in N_1 \mid \exists r \in R_{1\cap 2} \mid M(n_1) = (r, rl)\}$, and $cl_{1\cap 2} = \{n_2 \in N_2 \mid \exists r \in R_{1\cap 2} \mid M(n_2) = (r, cl)\}$ respectively.

Activity $a_{1;2} = (N_{1;2}, \rightarrow_{1;2})$ is then defined as follows:

$$
\begin{aligned}
N_{1;2} &= (N_1 \cup N_2) \backslash (cl_{1\cap 2} \cup rl_{1\cap 2}) \\
\rightarrow_{1;2} &= \{(n_i, n_j) \mid\ n_i \rightarrow_1 n_j \wedge\ n_j \notin rl_{1\cap 2}\} \cup \\
&\quad\ \{(n_i, n_j) \mid\ n_i \rightarrow_2 n_j \wedge\ n_i \notin cl_{1\cap 2}\} \cup \\
&\quad\ \{(n_1, n_2) \mid\ (\exists n_{rl} \in rl_{1\cap 2} \mid n_1 \rightarrow_1 n_{rl}) \wedge \\
&\qquad\qquad (\exists n_{cl} \in cl_{1\cap 2} \mid n_{cl} \rightarrow_2 n_2)\}.
\end{aligned}
$$

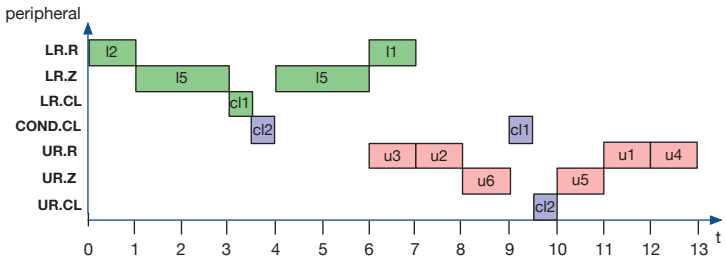


Figure 2.9: *Gantt chart of the execution of activity LR_PickFromCond;UR_PutOnCond depicted in Figure 2.8.*

This form of sequencing is similar to the notion of weak sequential composition [68], which is also defined relative to a dependency relation over a set of actions. Intuitively, given the set of shared resources, this operator removes intermediate release and claim nodes on these resources, and properly links the remaining dependencies. Notice that the result is again an activity satisfying Definition 2.1 and the corresponding consistency conditions.

Example 2.6 Consider activities LR_PickFromCond and UR_PutOnCond depicted in Figures 2.4 and 2.7 respectively. The result of their sequencing is shown in Figure 2.8. Notice that the result is again an activity satisfying the correct claiming and releasing of resources. This is achieved by replacing the release of the COND and CA resources of activity LR_PickFromCond and the respective claims in activity UR_PutOnCond with direct dependencies from (cl2,COND.CL) to (cl1,COND.CL) and from (l5,LR.Z) to (u3,UR.R). Its temporal execution is shown in Figure 2.9 and the makespan of the resulting activity is determined by mks(LR_PickFromCond;UR_PutOnCond) which yields the total of 13 time units.

2.4.2 Logistics Automata

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 sequence that satisfy these requirements is encoded by a *logistics automaton*.

Definition 2.9 — (Logistics automaton). A logistics automaton is a tuple $\langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$, where $\mathcal{S}$ is a finite (possibly empty) set of states, $\mathcal{A}ct$ is a finite (possibly empty) set of activities, $\dot{\rightarrow} \subseteq \mathcal{S} \times \mathcal{A}ct \times \mathcal{S}$ is a transition relation, and $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states, where $\mathcal{S}_0 = \emptyset$ if $\mathcal{S} = \emptyset$ and $\mathcal{S}_0 = \{s_0\}$ otherwise. Let $s \xrightarrow{a} s'$ be a shorthand for $(s, a, s') \in \dot{\rightarrow}$. The following additional properties must hold:

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

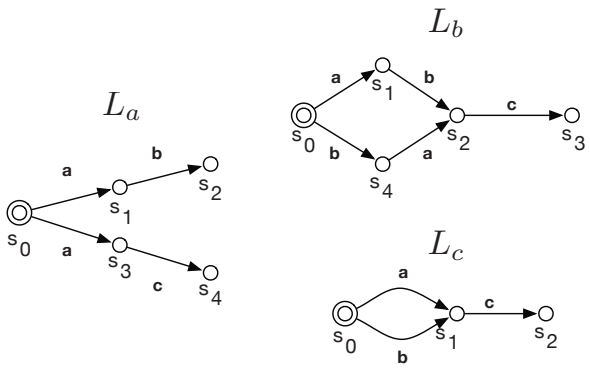


Figure 2.10: *Examples of valid logistics automata*

Example 2.7 Figure 2.10 depicts three examples, L_a, L_b and L_c, of valid logistics automata. Nodes represent states and edges represent transitions. Activities are annotated on edges and the initial state s_0 is distinguished by an extra circumference. Note that all automata satisfy the reachability and acyclicity properties as required. In contrast, Figure 2.11 depicts two examples, L_d and L_e, of invalid logistics automata. Example L_d does not satisfy acyclicity, since $s_0 \xrightarrow{b} s_1 \xrightarrow{a} s_0$ is a cycle; and L_e does not satisfy reachability, since s_4 is not reachable from s_0. States s_2 and s_4 of L_a have no outgoing edges; These states will be called final states.

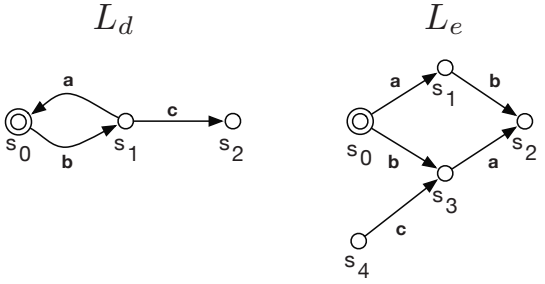


Figure 2.11: *Examples of invalid logistics automata*

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

Definition 2.10 — (Language of a logistics automaton). Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a logistics automaton. The language $\mathcal{L}(L)$ of L is defined by

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

Here $\mathcal{A}ct^*$ denotes the collection of all sequences of activities in $\mathcal{A}ct$. Each $\underline{a} \in \mathcal{A}ct^*$ is of the form $a_1 \ldots a_n$, 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 $\underline{a} = a_1 \ldots a_n \in \mathcal{A}ct^*$ we let $s \xrightarrow{\underline{a}} s'$ denote the existence of $s_1, \cdots, s_n \in \mathcal{S}$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n = s'$. Further $s \xrightarrow{\underline{a}}$ denotes that $s \xrightarrow{\underline{a}} s'$ for some $s' \in \mathcal{S}$, $s \dot{\rightarrow}$ denotes that $s \xrightarrow{a} s'$ for some $a \in \mathcal{A}ct$ and $s' \in \mathcal{S}$ and $s \not\dot{\rightarrow}$ denotes that $s \dot{\rightarrow}$ does not hold. Note that $\mathcal{L}(L) = \emptyset$ if $\mathcal{S} = \emptyset$ and $\mathcal{L}(L) = \{\epsilon\}$ if $\mathcal{S} = \{s_0\}$. Notice also that any sequence in the language should 'run to completion'. 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 2.8 Consider the logistic automata L_a, L_b and L_c depicted in Figure 2.10. Their respective languages are $\mathcal{L}(L_a) = \{a\ b, a\ c\}$, $\mathcal{L}(L_b) = \{abc, bac\}$ and $\mathcal{L}(L_c) = \{a\ c, b\ c\}$. Note that these languages are not prefix closed.

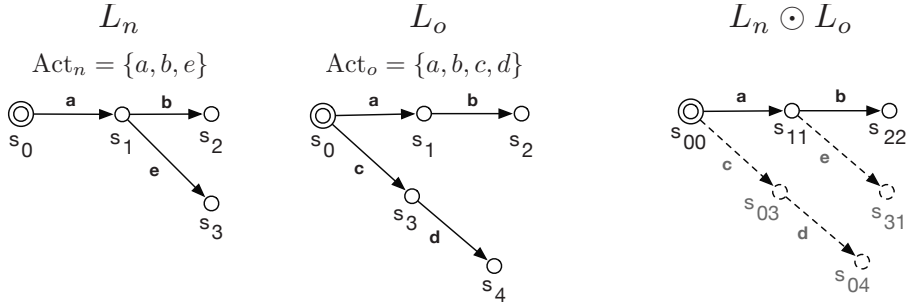


Figure 2.12: *Example of the composition operation with logistics automata L_n and L_o and the resulting composed automaton $L_n \odot L_o$. The dashed lines and circles show the pruned transitions and states after composition.*

2.4.3 Modular Logistics Specification

Even though a logistics automaton is able to encode the complete manufacturing of a batch of products, for large batch sizes or complex manufacturing jobs a monolithic automaton is not desired. In this work we take inspiration from the constraint-oriented specification style of the LOTOS framework [24] and the compositional specification of requirements in CIF [18] by defining a composition operator on logistics automata. The operator allows batches to be specified individually (i.e. each product flow can be specified by an individual automaton) and then composed to obtain an automata that encodes all the logistics requirements for the full batch of products.

Definition 2.11 — (Composition of Logistics Automata). Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ be logistics automata. Before we define the composition automaton $L_1 \odot L_2$, we first define relation $\dot{\rightarrow} \subseteq (\mathcal{S}_1 \times \mathcal{S}_2) \times (\mathcal{A}ct_1 \cup \mathcal{A}ct_2) \times (\mathcal{S}_1 \times \mathcal{S}_2)$ as the smallest set V satisfying the following inference rules:

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

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

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

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

$$\mathcal{S} = \begin{cases} \emptyset, & \text{if } \mathcal{S}_1 = \emptyset \text{ or } \mathcal{S}_2 = \emptyset \\ \{(s,t) \in \mathcal{S}_1 \times \mathcal{S}_2 \mid (s_{0_1}, s_{0_2}) \dot{\rightarrow}^* (s,t) \\ \text{ and for some } (s',t') \in \mathcal{S}_1 \times \mathcal{S}_2 \text{ with} \\ s' \not\rightarrow_1 \text{ and } t' \not\rightarrow_2, (s,t) \dot{\rightarrow}^* (s',t')\} & \text{if } \mathcal{S}_{0_1} = \{s_{0_1}\} \text{ and} \\ & \mathcal{S}_{0_2} = \{s_{0_2}\} \end{cases}$$

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

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

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

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

Example 2.9 Consider logistics automata L_n and L_o and the resulting composition $L_n \odot L_o$ depicted in Figure 2.12. The picture shows all reachable states in the product automaton, where the dashed circles and arrows with grayed labels are pruned. These pruned states and transitions do not lead to a state that corresponds both to a final state in L_n and to a final state in L_o.

Example 2.10 Consider logistics automata L_n and L_p and the resulting composition $L_n \odot L_p$ depicted in Figure 2.13. In this case all reachable states and transitions are pruned, resulting in an empty automaton.

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

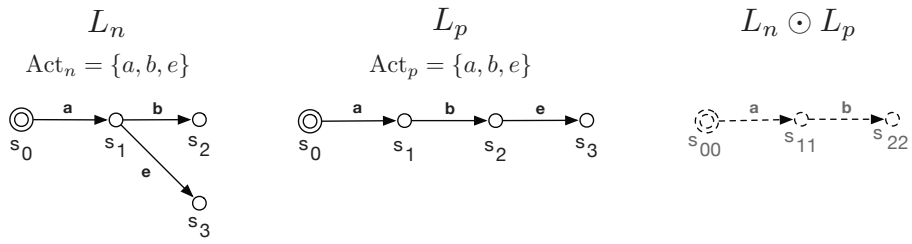


Figure 2.13: *Example of an empty automaton as a result of the composition of logistics automata L_n and L_p.*

Lemma 2.1 Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ be logistics automata. Then $\mathcal{L}(L_1 \odot L_2) = \{\underline{a} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^* \mid \underline{a} \backslash \mathcal{A}ct_1 \in \mathcal{L}(L_1)$ 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.

Proof. Let $(s_1, s_2), (s_1', s_2') \in \mathcal{S}_1 \times \mathcal{S}_2$ and $\underline{a} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^*$. Then by induction on the structure of $\underline{a}$ and by distinguishing the different inference rules of Definition 2.11 it is not hard to show that $(s_1, s_2) \stackrel{\underline{a}}{\dot{\rightarrow}}'_{L_1 \odot L_2} (s_1', s_2')$ iff $s_1 \xrightarrow{\underline{a} \backslash \mathcal{A}ct_1}_1 s_1'$ and $s_2 \xrightarrow{\underline{a} \backslash \mathcal{A}ct_2}_2 s_2'$, where $\dot{\rightarrow}'_{L_1 \odot L_2}$ refers to the transition relation on the product state-space *before pruning* (see Definition 2.11).

Now let $\underline{a} \in \mathcal{L}(L_1 \odot L_2)$. Then $(s_{0_1}, s_{0_2}) \stackrel{\underline{a}}{\Rightarrow}_{L_1 \odot L_2} (s_1, s_2)$ for some $(s_1, s_2) \in \mathcal{S}_{L_1 \odot L_2}$ such that $(s_1, s_2) \not\dot{\rightarrow}$. But then $s_1 \not\dot{\rightarrow}_1$ and $s_2 \not\dot{\rightarrow}_2$. Further $(s_{0_1}, s_{0_2}) \stackrel{\underline{a}}{\dot{\rightarrow}}'_{L_1 \odot L_2} (s_1, s_2)$ so $s_{0_1} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_1}_1 s_1$ and $s_{0_2} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_2}_2 s_2$. Hence $\underline{a} \backslash \mathcal{A}ct_1 \in \mathcal{L}(L_1)$ and $\underline{a} \backslash \mathcal{A}ct_2 \in \mathcal{L}(L_2)$. Hence $\underline{a} \in \{\underline{a} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^* \mid \underline{a} \backslash \mathcal{A}ct_1 \in \mathcal{L}(L_1)$ and $\underline{a} \backslash \mathcal{A}ct_2 \in \mathcal{L}(L_2)\}$.

Vice versa, let $\underline{a} \in \{\underline{a} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^* \mid \underline{a} \backslash \mathcal{A}ct_1 \in \mathcal{L}(L_1)$ and $\underline{a} \backslash \mathcal{A}ct_2 \in \mathcal{L}(L_2)\}$. Then $s_{0_1} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_1}_1 s_1$ and $s_{0_2} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_2}_2 s_2$ for some $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$ for which $s_1 \not\dot{\rightarrow}_1$ and $s_2 \not\dot{\rightarrow}_2$ (where $s_{0_1} \in S_{0_1}$ and $s_{0_2} \in S_{0_2}$). We then have $(s_{0_1}, s_{0_2}) \stackrel{\underline{a}}{\dot{\rightarrow}}'_{L_1 \odot L_2} (s_1, s_2)$. By Definition 2.11, $(s_{0_1}, s_{0_2}) \in \mathcal{S}_{0_{L_1 \odot L_2}}$ and $(s_1, s_2) \in \mathcal{S}_{L_1 \odot L_2}$ and thus $(s_{0_1}, s_{0_2}) \stackrel{\underline{a}}{\Rightarrow}_{L_1 \odot L_2} (s_1, s_2)$. Therefore $\underline{a} \in \mathcal{L}(L_1 \odot L_2)$. ■

Example 2.11 Consider logistics automata L_n, L_o and $L_n \odot L_o$ of Figure 2.12. Notice that the languages of L_n and L_o are $\mathcal{L}(L_n) = \{a\ b, a\ e\}$ and $\mathcal{L}(L_o) = \{a\ b, c\ d\}$. Notice that only sequence ab satisfies $ab \backslash \mathcal{A}ct_n \in \mathcal{L}(L_n)$ and $ab \backslash \mathcal{A}ct_o \in \mathcal{L}(L_o)$, and therefore by Lemma 2.1, $\mathcal{L}(L_n \odot L_o) = \{a\ b\}$ which is consistent with Figure 2.12.

2.4.4 Constraint Automata

We have previously introduced the concept of logistics automaton. With such automata we capture the product flow of a product and ensure the completion of its manufacturing. In addition we express constraints within the same product, concerning for instance a safe handover between resources. To specify the behavior of a batch of products in a modular way we introduced the composition operator. 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. We express such constraints in terms of *constraint automata* and introduce a *constraint operator* to compose them with logistics automata.

Definition 2.12 — (Constraint automaton). A constraint automaton is a tuple $\langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$, where $\mathcal{S}$ is a finite (possibly empty) set of states, $\mathcal{A}ct$ is a finite (possibly empty) set of activities, $\dot{\rightarrow} \subseteq \mathcal{S} \times \mathcal{A}ct \times \mathcal{S}$ is a transition relation, and $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states, where $\mathcal{S}_0 = \emptyset$ if $\mathcal{S} = \emptyset$ and $\mathcal{S}_0 = \{s_0\}$ otherwise. The following additional property must hold:

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

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

Definition 2.13 — (Language of a constraint automaton). Let $C = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a constraint automaton. The language $\mathcal{L}(C)$ of C is defined by

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

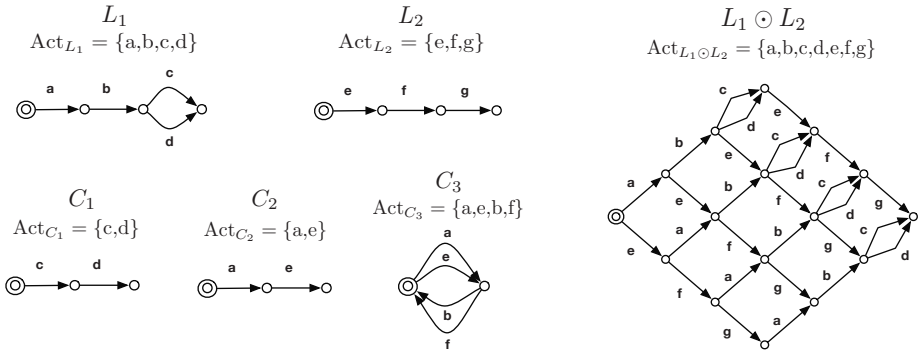


Figure 2.14: *Example of different constraint automata C_1, C_2 and C_3 referring to different ordering requirements on logistics automata L_1 and L_2 and logistics automaton $L_1 \odot L_2$.*

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.

Example 2.12 Consider logistics automata L_1, L_2 and $L_1 \odot L_2$ and constraint automata C_1, C_2 and C_3 depicted in Figure 2.14. 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 $L_1 \odot L_2$. C_2 imposes an input order constraint, i.e. activity a must always precede activity e. C_3 imposes an alternating execution of activities b or f with activities a or e. For example, for every instance of activities a or e an instance of activities b or f must follow before another instance of a or e is executed. For this reason C_3 is a recursive constraint automaton encoding an infinite language.

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

Definition 2.14 — (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 $L \restriction C$ 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:

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

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

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

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

Finally, the constrained automaton $L \restriction C$ is 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. However, note 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

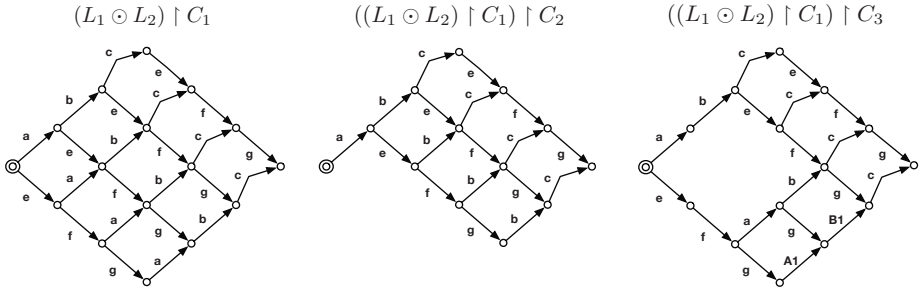


Figure 2.15: *Example showing the constraining of automaton $L_1 \odot L_2$ using the different constraint automata C_1, C_2 and C_3 depicted in Figure 2.14.*

completion of the different products in a batch). On the contrary, the composition operator requires that both logistics automata run to completion. It was not strictly necessary to add the concepts of constraint automaton and constraint operator since constraints can in principle be encoded as logistics automata (by unfolding recursive loops sufficiently often). However, the automaton obtained by composition would in many cases have a larger state-space than the constrained automaton, which is something we try to prevent. In addition, logistics automata encoding constraints would be incomprehensible because of their size and would also be less reusable. For these reasons we decided to introduce constraint automata and the constraint operator.

Example 2.13 Figure 2.15 shows the application of constraints C_1, C_2 and C_3 to automaton $L_1 \odot L_2$ (of Figure 2.14). Since C_1 requires activity d to be preceded by c, $(L_1 \odot L_2) \upharpoonright C_1$ does not contain d transition anymore. The application of constraint C_2 to $(L_1 \odot L_2) \upharpoonright C_1$ further removes all behavior in which activity e is not preceded by a. When C_3 is applied to $(L_1 \odot L_2)$ 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 state-space has reduced in each of the three cases. This does not hold in general however. In Chapter 4 we will establish the conditions which guarantee that constraining does not increase the state-space.

The language of a constrained logistics automaton can be computed from the languages of its constituent automata, basically by filtering out activity sequences that are not consistent with the constraint:

Lemma 2.2 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

$$\mathcal{L}(L \restriction C) = \{\underline{a} \in \mathcal{L}(L) \mid \underline{a} \backslash \mathcal{A}ct_2 \in \mathcal{L}(C)\}$$

Proof. Let $(s_1, s_2), (s_1', s_2') \in \mathcal{S}_1 \times \mathcal{S}_2$ and $\underline{a} \in \mathcal{A}ct_1^*$. By induction on the structure of $\underline{a}$ and by distinguishing the different inference rules of Definition 2.14 it can be shown that $(s_1, s_2) \xrightarrow{\underline{a}}{}'_{L \restriction C} (s_1', s_2')$ iff $s_1 \xrightarrow{\underline{a}}_1 s_1'$ and $s_2 \xrightarrow{\underline{a} \backslash \mathcal{A}ct_2}_2 s_2'$, where $\dot{\rightarrow}'_{L \restriction C}$ refers to the transition relation on the product state-space before pruning (see Definition 2.14).

Now let $\underline{b} \in \mathcal{L}(L \restriction C)$. Then $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{b}}_{L \restriction C} (s_1, s_2)$ for some $(s_1, s_2) \in \mathcal{S}_{L \restriction C}$ such that $(s_1, s_2) \not\dot{\rightarrow}_{L \restriction C}$. But then also $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{b}}{}'_{L \restriction C} (s_1, s_2)$ and thus $s_{0_1} \xrightarrow{\underline{b}}_1 s_1$ and $s_{0_2} \xrightarrow{\underline{b} \backslash \mathcal{A}ct_2}_2 s_2$ and $s_1 \not\dot{\rightarrow}_1'$. Hence $\underline{b} \in \mathcal{L}(L)$ and $\underline{b} \backslash \mathcal{A}ct_2 \in \mathcal{L}(C)$ and therefore $\underline{b} \in \{\underline{a} \in \mathcal{L}(L) \mid \underline{a} \backslash \mathcal{A}ct_2 \in \mathcal{L}(C)\}$.

Vice versa, let $\underline{b} \in \{\underline{a} \in \mathcal{L}(L) \mid \underline{a} \backslash \mathcal{A}ct_2 \in \mathcal{L}(C)\}$. Then $s_{0_1} \xrightarrow{\underline{a}} s_1$ and $s_{0_2} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_2}_2 s_2$ for some $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$ for which $s_1 \not\dot{\rightarrow}_1$. But then $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{b}}{}'_{L \restriction C} (s_1, s_2)$ and since $s_1 \not\dot{\rightarrow}_1$ we further know from Definition 2.14 that $(s_{0_1}, s_{0_2}), (s_1, s_2) \in \mathcal{S}_{L \restriction C}$. Hence $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{b}}_{L \restriction C} (s_1, s_2)$, $(s_1, s_2) \not\dot{\rightarrow}_{L \restriction C}$ and therefore $\underline{b} \in \mathcal{L}(L \restriction C)$. ■

The application of a constraint to a logistics automaton, results in a subset of the original language. This follows immediately from Lemma 2.2:

Lemma 2.3 — (Language constraining). Let L be a logistics automaton and let C be a constraint on L. Then $\mathcal{L}(L \restriction C) \subseteq \mathcal{L}(L)$.

Example 2.14 Consider Figure 2.15. It is easy to see that $\mathcal{L}(((L_1 \odot L_2) \restriction C_1) \restriction C_2) \subseteq \mathcal{L}((L_1 \odot L_2) \restriction C_1)$ (consistent with Lemma 2.3). Consider further activity sequence $a\ b\ c\ e\ f\ g$. Clearly this sequence is part of $\mathcal{L}((L_1 \odot L_2) \restriction C_1)$. Further $a\,b\,c\,e\,f\,g \backslash \{a, e\} = a\,e$ and $a\,e \in \mathcal{L}(C_2)$. Thus from Lemma 2.2 it follows that also $a\,b\,c\,e\,f\,g \in \mathcal{L}(((L_1 \odot L_2) \restriction C_1) \restriction C_2)$.

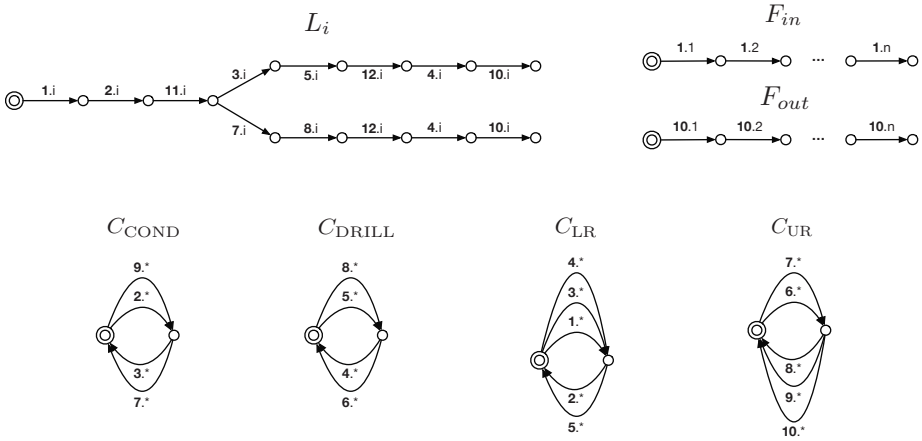


Figure 2.16: *Requirements and constraints of the batch logistics specification of the Twilight System.*

2.4.5 Application to the Twilight

To specify the logistics of the Twilight system we need to consider both logistics requirements and system constraints. Figure 2.16 depicts several automata that describe these requirements and constraints for the Twilight system for a batch of n products. We assume that each product in the batch is associated with a copy of the Twilight activities listed in Table 2.3. This is indicated by an alias **activity**:*product* in the automata of Figure 2.16. For example **2**.i refers to the copy of activity **2** of Table 2.3 associated with product i. We will start by first introducing the logistics requirements and then adding the necessary system constraints to ensure that the final logistics automaton encodes only activity sequences that capture the complete manufacturing of a batch of products which satisfies all system constraints.

Figure 2.16 (a) shows the logistics requirements for an individual product i of a batch of n products (where $1 \leq i \leq n$) modeled as logistics automaton L_i, where i indicates the product number. The product flow follows the explanation described in Section 2.1.3. Note that the product flow accounts for the choice of the LR or UR as the resource responsible for moving a product from the COND to the DRILL. The remaining automata reflect the system requirements described in Section 2.1.5. Constraint automata F_{in} and F_{out} of Figure 2.16 (b) describe the First-In and First-Out (FIFO) requirements, where we enforce

that products must be outputted in the same order as they where inputted. Finally, constraint automata C_{COND},C_{DRILL}, C_{LR} and C_{UR} depicted in Figure 2.16 (c),(d),(e) and (f) represent the capacity constraints of the corresponding resources. To avoid cluttering the automata figures we write $\mathbf{a}.*$ to represent n different transitions with labels $\mathbf{a}.1, \cdots, \mathbf{a}.n$ (where **a** denotes the name of an activity). This implies that if this transition is enabled any activity **a** on behalf of any wafer i in the batch is enabled. The capacity of a resource is modeled by a two state constraint automaton. We assume the resource of the twilight to start *empty* and to be of unary capacity. Therefore the initial state of a capacity constraint allows a transition for every activity that *occupies* the resource. Once a resource is *occupied*, only transitions that *empty* the resource are allowed. Since we want the behavior to be continuously alternating between *empty* and *occupied* we capture these requirements using recursive constraint automata. Consider the case of the LR (Figure 2.16 (e)). The resource starts empty so activities which pick a product from other resources such as **1**.i (LR_PickFromInput), **3**.i (LR_PickFromCond) and **4**.i (LR_PickFromDrill) are enabled. Once any of these activities is executed the automaton transits to a state for which only activities which place a product on a different resource are enabled, such as **2**.i (LR_PutOnCond) and **5**.i (LR_PutOnDrill).

To derive the logistics automaton of a batch of products we can use the composition operator. Consider we have a batch of two products ($n = 2$) where the individual requirements of each product by two copies of automata L_i (Figure 2.16), L_1 and L_2. Figure 2.17 depicts logistics automaton $L_1 \odot L_2$ capturing the logistics requirements of the batch of two products. Any activity sequence of $L_1 \odot L_2$ captures the manufacturing of two products in the Twilight system. For readability, activities performed by product 1 and 2 are colored in orange and blue respectively. Notice that we sequenced activities **7**.i;**8**.i (UR_PickFromCond and UR_PutOnDrill) and **3**.i;**5**.i (LR_PickFromCond and LR_PutOnDrill), which reflect the choice between the LR or the UR moving a product from the COND to the DRILL. We have done so for readability since distinguishing all the interleaving possibilities would have made the figure too complicated to be visualized. In the next chapter when we optimize the Twilight system we take the full specification into account.

Even though any sequence in $L_1 \odot L_2$ captures the logistics requirements for two products in the Twilight system, certain sequences might not satisfy all system constraints as listed in Section 2.1.5. Therefore we need to constrain

$L_1 \odot L_2$ with the Twilight constraints depicted in Figure 2.16. To ensure FIFO ordering we constrain with F_{in} and F_{out} to obtain $(L_1 \odot L_2) \restriction F_{in} \restriction F_{out}$. Then adding the capacity constraints we obtain $\big((L_1 \odot L_2) \restriction F_{in} \restriction F_{out}\big) \restriction C_{\text{COND}} \restriction C_{\text{DRILL}} \restriction C_{\text{LR}} \restriction C_{\text{UR}}$. Doing so results in the automaton depicted in Figure 2.18. In this case any sequence of activities implies the manufacturing of two products while satisfying all system constraints.

Notice that many activity sequences of $L_1 \odot L_2$ are no longer possible. For instance, any activity sequence that starts with activity **1**.2 (LR_PickFromInput) is disallowed since it violates FIFO ordering. Another example is the removal of activity sequences where activities **5**.i (LR_PutOnDrill) or **2**.i (LR_PutOnCond) happen without activities **4**.i (LR_PickFromDrill), **3**.i (LR_ PickFromCond) or **1**.i (LR_PickFromInput) occurring in between. These are removed to satisfy the capacity constraint of the LR resource encoded by C_{LR}. Intuitively, $L_1 \odot L_2$ (Figure 2.17) represents all 'possible' activities sequences of a two product batch while $\big((L_1 \odot L_2) \restriction F_{in} \restriction F_{out}\big) \restriction C_{\text{COND}} \restriction C_{\text{DRILL}} \restriction C_{\text{LR}} \restriction C_{\text{UR}}$ (Figure 2.18) represents all 'allowed' activity sequences.

2.5 Related Work

The specification approach presented in this chapter is an effort of combining different modeling solutions in a way that fits the manufacturing systems domain and provides a solid foundation for their design exploration and optimization. The goal is to achieve a clear separation of concerns between functional and temporal aspects of a system. Furthermore, we have aimed for a specification framework that is modular and compositional in the sense that different system components and requirements are described in modules which can later be composed to describe the full system facilitating complex system specification. Each module can be individually validated against functional requirements and it is guaranteed that these requirements are still satisfied after the composition. This specification approach combines ingredients form the domains of of Scenario-Aware Data Flow (SADF) [39] and Supervisory Control Theory (SCT) [66]. The scenario-based modeling approach of SADF provides the motivation for the definition of activities as end-to-end deterministic pieces of functional system behavior. On top of the scenario definitions, a Finite-State Machine can be specified to dictate the allowed ordering of scenario sequences much like our concept of a logistics automaton. An SADF scenario is more general, since it can also contain cycles. In one of our earliest works [15] we

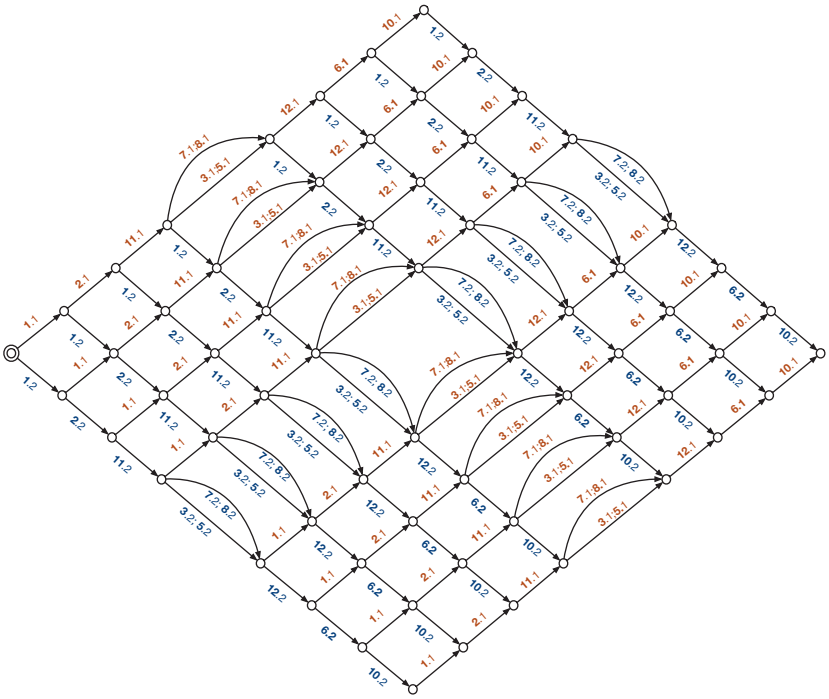


Figure 2.17: *Logistics automaton of a batch of two products of the Twilight System. The automaton is obtained by the composition of two individual product logistics requirements* $L_1 \odot L_2$.

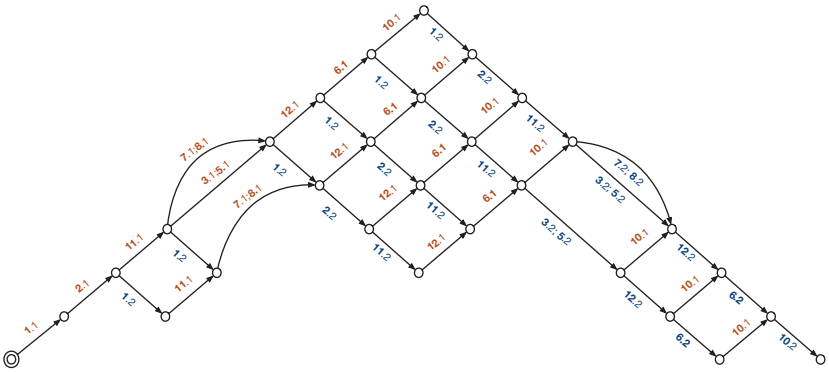


Figure 2.18: *Logistics automaton of batch of two products after the composition with FIFO constraints and constraining with Capacity constraint* $\left(\left((L_1 \odot L_2) \restriction F_{in} \restriction F_{out}\right) \restriction C_{COND} \restriction C_{DRILL} \restriction C_{LR} \restriction C_{UR}\right)$.

explore FSM-SADF models in an attempt to model different operational scenarios of a manufacturing system by deriving an FSM that is able to capture the resource sharing and resource assignment dynamics of manufacturing systems. This work led to many of the concepts and methods presented in this chapter and in [70]. It is possible to convert a specification model in our framework to an FSM-SADF model. Each activity can be mapped onto an SADF scenario and the logistics automaton of the system corresponds to a finite-state machine in FSM-SADF.

On the modular specification of logistics, a similar approach and source of inspiration is the one of Supervisory Control Theory (SCT) which is applied in [71] for the modular specification and synthesis of manufacturing systems controllers. Further applications of SCT can be found in [19], [36], [80]. Our composition and constraining operator are inspired by the synchronous product and synthesis steps in SCT theory. Compared to SCT, our approach restricts the specification expressiveness to better suit the manufacturing domain and focuses on makespan analysis. We implement our logistics specification approach and methods using the CIF3 tooling [18], which we discuss further in Chapter 3. However, in our models we focus on good-weather behavior and thus do not include uncontrollable events as is the case of SCT. Furthermore, we only focus on the reachability of final states in logistics automata as a liveness properties. In comparison, SCT provides the concept of marked states and marked languages which we do not use in our automata definitions. A further comparison of the methodologies including the temporal optimization is found in Chapter 3.

2.6 Conclusions

This chapter has introduced the approach and concepts for the specification of manufacturing systems. We showed how to decompose a manufacturing system into a plant by defining resources, peripherals and peripheral actions. Moreover, we showed how to capture deterministic pieces of functional behavior as activities by using the building blocks of the plant specification. Different levels of granularity of the behavior captured by an activity might be chosen for different purposes (e.g. functional verification of requirements or optimization). To aid this, the sequencing operator is provided such that larger activities can be created by the sequencing of smaller ones, to the point that an activity captures the complete behavior necessary to manufacture a product.

On top of the set of activities of a system this chapter also discussed the specification of activity sequences within a manufacturing system. This is captured as a logistics automaton which language encodes the set of possible activity sequences. We showed how the logistics automaton of a batch of products can be modularly specified by describing individual logistics requirements as individual automata and by using the composition operator. Furthermore, we introduced the concept of constraint automata to capture system constraints (i.e. input/output ordering constraints, capacity constraints and safety constraints). These can also be specified in a modular way and composed with logistics automata using the constraint operator. The concepts of logistics and constraint automata together with the composition and constraint operator defines our modular specification of the logistics of flexible manufacturing systems. Such a modular approach allows us to deal with the specification and validation of complex flexible manufacturing systems. This will be further illustrated in Chapter 6. Furthermore this modularity is further explored in Chapter 4 to bound and prune the optimization-space of flexible manufacturing systems.

3 | Optimization of Flexible Manufacturing Systems

In the previous chapter we discussed the specification of flexible manufacturing systems. We showed how to decompose a system *plant* into sets of resources, peripherals and actions, and capture the functional behavior of the system in terms of *activities* and a language of *activity sequences* encoded by a *logistics automaton*. The overall goal of the framework is to explore the performance of different designs of flexible manufacturing systems, for instance by considering different layouts and system resources. A fundamental step in that process is the productivity analysis of the system design in study. Therefore in this chapter we discuss the *optimization* of a such a specification in order to find the optimal makespan for a given batch of products.

We start by defining the *Batch Makespan Optimization* (BMO) problem to find the activity sequence with the lowest makespan within the language of activity sequences of a logistics automaton. This activity sequence minimizes the completion time of a batch of products. Figure 3.1 depicts the different concepts as well as the necessary analysis techniques to find a solution to the BMO problem. We start by translating the structural and timing information of activities to *(max,+) matrices*. By the means of a *(max,+) expansion* algorithm we annotate the logistics automaton with the temporal characterization of activities to construct a *(max,+) automaton*. When constructing the (max,+) automaton the different temporal behavior of activities might lead to multiplication of states of the logistics automaton. This multiplication occurs due to the addition of a resource time-stamp vector to each state in the (max,+) automaton. This time-stamp vector contains the availability times of all system resources after the execution of the activity sequence leading to that state.

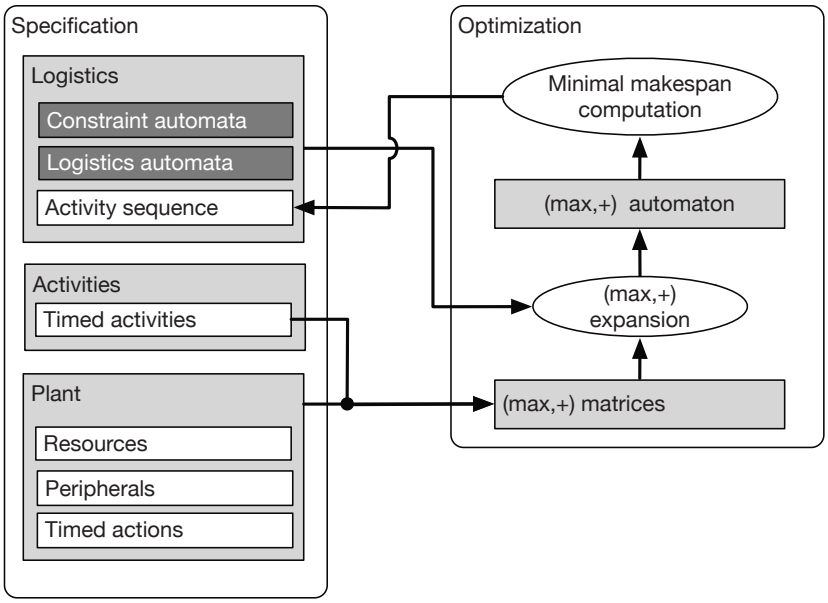


Figure 3.1: *Overview of the framework concepts for the* Specification *and* Optimization *domains.*

Each activity sequence in the language of the (max,+) automaton captures the manufacturing of a batch of products for which the completion time can be obtained by computing the the resource time-stamp vector of the final states (states with no outgoing transitions). Therefore the solution to the BMO problem can be obtained by exploring the state-space of the (max,+) automaton, which we will therefore also denote as *optimization-space*. In this chapter we show all the necessary steps to compute and explore this optimization-space to find a solution to the BMO problem. The Twilight system is used to illustrate these steps.

The chapter is organized as follows. Section 3.1 introduces the BMO problem. Section 3.2 discusses the (max,+) characterization of activities and activity sequences. Section 3.3 introduces the concept of (max,+) automaton and explains the exploration of the optimization-space to find a solution to the BMO problem. Section 3.4 uses the introduced concepts to optimize the Twilight system specification. Section 3.5 discusses the related work and finally Section 3.6 concludes the chapter.

3.1 Batch Makespan Optimization

Two common metrics to express the productivity of manufacturing system designs are *throughput* (number of products produced per time unit) and *makespan* (the completion time of a batch of products) [51], [61], [82]. Throughput analysis focuses on the analysis of the steady-state output of a system, while makespan analysis considers the total elapsed time from start to finish of a particular batch of products. Flexible manufacturing systems often work with small batches of products and with mixed product flows (manufacturing of different product types) for which steady-state behaviors are not so interesting to study [48], [61]. It is therefore valuable to look at batch-oriented aspects and study the makespan of the system to evaluate the productivity impact of different logistical choices. In this thesis we focus on the makespan optimization of a manufacturing system and define the Batch Makespan Optimization (BMO) problem.
The goal is to determine an activity sequence that leads to the lowest makespan for a specific batch of products. In our framework we specify a *language of activity sequences* where each activity sequence in the language of a logistics automaton captures the complete and correct manufacturing of a batch of products. Therefore we define our BMO problem in terms of the language of activities sequences of a logistics automaton, as follows:

Problem 3.1 — (Batch Makespan Optimization). Given a Logistics automaton L determine an $\underline{a} \in \mathcal{L}(L)$ such that

$$mks(\underline{a}) \leq mks(\underline{a}')$$

for all $\underline{a}' \in \mathcal{L}(L)$. Here we let $mks(\underline{a})$ denote $mks(a_1; ...; a_n)$, when $\underline{a} = a_1\ a_2\ \cdots\ a_n$.
By language inclusion we can establish sub-optimal solutions to the BMO problem and find bounds on the optimal makespan. This is posed in the following Lemma.

Lemma 3.1 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 $mks(\underline{a}) \geq mks(\underline{a}')$.

In the remainder of this chapter we discuss the individual steps taken to find a solution to the BMO problem.

3.2 Activities as (max,+) Matrices

In the previous chapter we showed that we can capture the temporal behavior of an activity a by computing the ASAP *start* and *end* times of all its actions assuming a given initial resource time-stamp vector γ_R. Furthermore, we showed that using this timing information we can compute the updated resource time-stamp vector after executing activity a as $update(\gamma_R, a)$ (Definition 2.6). In this section we show that we can capture the timing behavior of an activity and compute $update(\gamma_R, a)$ using (max,+) algebra [10], [41]. We use these conclusions later in this chapter in our approach to solve the BMO problem.

3.2.1 (max,+) Algebra

This section introduces (max,+) algebra based on the summary of [70]. Recall that the two essential characteristics of the temporal execution of an activity are *synchronization* (a node waits for all its incoming dependencies to finish) and *delay* (a node takes a fixed amount of time to execute). These can be matched to the $(\max, +)$ operators *max* and *addition*, defined over the set $\mathbb{R}^{-\infty} = \mathbb{R} \cup \{-\infty\}$. The $\max$ and $+$ operators are defined as in a usual algebra, with the additional convention that $-\infty$ is the unit element of $\max$: $\max(-\infty, x) = \max(x, -\infty) = x$, and the zero-element of addition: $-\infty + x = x + -\infty = -\infty$. Addition distributes over the max operator, i.e. $x + \max(y, z) = \max(x + y, x + z)$.

Since $(\max, +)$ algebra is a linear algebra, it can be easily extended to matrices and vectors. Given matrix $\boldsymbol{A}$ and vector $\boldsymbol{x}$, we use $\boldsymbol{A} \otimes \boldsymbol{x}$ to denote the $(\max, +)$ matrix multiplication. Given $m \times p$ matrix $\boldsymbol{A}$ and $p \times n$ matrix $\boldsymbol{B}$, the elements of the resulting matrix $\boldsymbol{A} \otimes \boldsymbol{B}$ are determined by: $[A \otimes B]_{ij} = \max_{k=1}^{p}([A]_{ik} + [B]_{kj})$. For any vector $\boldsymbol{x}$, $\|\boldsymbol{x}\| = \max_i x_i$ denotes the vector norm of $\boldsymbol{x}$. We use $\mathbf{0}$ to denote a vector with all zero-valued entries.

3.2.2 (max,+) Activity Semantics

The temporal behavior of an activity can be captured by a single (max,+) matrix which encodes the critical timing dependencies between the different resource claims and releases of an activity. We can then compute the updated resource time-stamp vector after the execution of an activity by multiplying the corresponding (max,+) matrix with the initial resource time-stamp vector. This is illustrated in the following example:

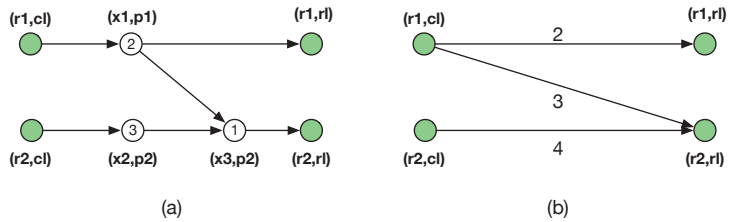


Figure 3.2: *(a) example activity* a *and (b) longest paths for every pair of resource of activity* a.

Example 3.1 Consider activity a depicted in Figure 3.2 (a). Assume $\mathcal{R} = \{r_1, r_2\}$. Note that the activity uses both resources r_1 and r_2 and that $R(p_1) = r_1$ and $R(p_2) = r_2$ and further employs three actions $x1, x2$ and $x3$ for which $T(x1) = 2$, $T(x2) = 3$ and $T(x3) = 1$. Consider symbolic resource time function γ_R. We will represent it as a resource-time stamp vector $\underline{\gamma_R} = [\gamma_R(r_1), \gamma_R(r_2)]^\intercal$ (by assuming that the first entry in the vector represents r_1 and the second entry in the vector represents r_2). Following Definition 2.5 the *end* times of all nodes are computed as follows:

$$
\begin{aligned}
end((r_1, cl)) &= \gamma_R(r_1) \\
end((r_2, cl)) &= \gamma_R(r_2) \\
end((x1, p1)) &= \max(end((r_1, cl))) + T((x1, p1)) = \gamma_R(r_1) + 2 \\
end((x2, p2)) &= \max(end((r_2, cl))) + T((x2, p2)) = \gamma_R(r_2) + 3 \\
end((x3, p2)) &= \max(end((x1, p1)), end((x2, p2)) + T((x3, p2)) \\
&= \max(\gamma_R(r_1) + 2, \gamma_R(r_2) + 3) + 1 \\
&= \max(\gamma_R(r_1) + 3, \gamma_R(r_2) + 4) \\
end((r_1, cl)) &= end((x1, p1)) = \gamma_R(r_1) + 2 \\
end((r_2, cl)) &= end((x3, p2)) = \max(\gamma_R(r_1) + 3, \gamma_R(r_2) + 4).
\end{aligned}
$$

We can now write the (max,+) characterization of $end((r1, cl))$ and $end((r2, cl))$ to obtain:

$$
\begin{aligned}
end((r1, rl)) &= \max(\gamma_R(r_1) + 2, \gamma_R(r_2) + -\infty) \\
end((r2, rl)) &= \max(\gamma_R(r_1) + 3, \gamma_R(r_2) + 4).
\end{aligned}
$$

So by Definition 2.6 we have $update(\gamma_R, a) = [\max(\gamma_R(r_1) + 2, \gamma_R(r_2) + -\infty), \max(\gamma_R(r_1) + 3, \gamma_R(r_2) + 4)]^\intercal$. This updated vector can however also be obtained by characterizing activity a by (max,+) matrix $\boldsymbol{M}_a$

$$\boldsymbol{M}_a = \begin{bmatrix} 2 & -\infty \\ 3 & 4 \end{bmatrix}$$

and computing $\boldsymbol{M}_a \otimes \underline{\gamma_R}$ as

$$\begin{aligned} update(\gamma_R, a) &= \boldsymbol{M}_a \otimes \underline{\gamma_R} \\ &= \begin{bmatrix} 2 & -\infty \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} \gamma_R(r_1) \\ \gamma_R(r_2) \end{bmatrix} \\ &= \begin{bmatrix} \max(2 + \gamma_R(r_1), -\infty + \gamma_R(r_2)) \\ \max(3 + \gamma_R(r_1), 4 + \gamma_R(r_2)) \end{bmatrix} \end{aligned}$$

We thus have that $update(\underline{\gamma_R}, a) = \boldsymbol{M}_a \otimes \underline{\gamma_R}$. If we assume the initial resource time-stamp vector to be $\underline{\mathbf{0}_R}$ then:

$$\begin{aligned} update(\underline{\mathbf{0}_R}, a) &= \boldsymbol{M}_a \otimes \underline{\mathbf{0}_R} \\ &= \begin{bmatrix} 2 & -\infty \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \max(2 + \gamma_R(r_1), -\infty + \gamma_R(r_2)) \\ \max(3 + \gamma_R(r_1), 4 + \gamma_R(r_2)) \end{bmatrix} \\ &= \begin{bmatrix} \max(2 + 0, -\infty + 0) \\ \max(3 + 0, 4 + 0) \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \end{aligned}$$

Now, if we assume a different initial resource time-stamp we may obtain a different resource time-stamp vector. For instance if $\underline{\gamma_R} = [2, 0]^T$ then:

$$\boldsymbol{M}_a \otimes \underline{\gamma_R} = \begin{bmatrix} 2 & -\infty \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

Figure 3.3 depicts the Gantt chart execution of activity *a* with respect to these different initial vectors. Note that the different initial resource

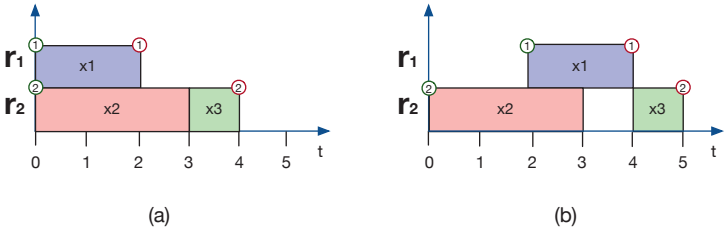


Figure 3.3: *Gantt charts of the temporal execution of activity* a *of Figure 3.2 assuming (a)* $\underline{\mathbf{0}_R}$ *and (b)* $\underline{\gamma_R} = [2, 0]^T$.

time-stamp vectors indeed lead to different temporal executions of *a* and furthermore that the different temporal aspects of *a* are all captured by the same (max,+) matrix characterization. The (max,+) characterization abstracts from the individual start times and execution times of actions. In fact the only observable temporal behavior are the entries of the initial and updated resource time-stamp vectors, shown in Figure 3.3 by the small circles with the resource number written within. The gap of 1 time unit between actions b and c is not observable. For such details we would still need to rely on the computation of all *start* and *end* times of actions as explained in Section 2.3.2.

It is a well-know that (max,+) algebra can be used to mathematically describe timed synchronous systems [10], [40], [41]. This observation also applies to activities, as illustrated in Example 3.1. Therefore we pose without proof the following lemma generalizing the computation of the updated resource time-stamp vector.

Lemma 3.2 — ((max,+) based update computation). Given an activity a and an initial resource time-stamp vector $\underline{\gamma_R}$ then:

$$\underline{update(\gamma_R, a)} = \boldsymbol{M}_a \otimes \underline{\gamma_R}$$

In the sequel, for readability reasons, we will not explicitly distinguish resource time-stamp function from resource time-stamp vectors anymore. Hence we will write $update(\gamma_R, a) = \boldsymbol{M}_a \otimes \gamma_R$ from now on.

Computing the (max,+) matrix of an activity

Algorithm 1 describes the computation of (max,+) matrices from activities. Algorithm 1 runs in polynomial time. It assumes as input an activity a, the set

Algorithm 1 Compute the Max-Plus matrix of an activity

```
1:  procedure ComputeMaxPlusMatrix(a, R(a), R)
2:      Assume resource time stamp vector 0_R
3:      i = 1
4:      for r_i ∈ R do
5:          j = 1
6:          for r_j ∈ R do
7:              if r_i ∈ R(a) and r_j ∈ R(a) then
8:                  M(i, j) = longestPath(a, r_j, r_i)
9:              else
10:                 if i = j then
11:                     M(i, j) = 0
12:                 else
13:                     M(i, j) = −∞
14:             j++
15:         i++
16:     return M
```

of resources used by the activity $\mathcal{R}(a)$ and the set of resources $\mathcal{R}$. The output is a matrix of size $n \times n$ where n corresponds to the total number of elements in $\mathcal{R}$ of the specified system plant, even if the activity only uses a subset of the plant resources. Further, it assumes that the resource elements in $\mathcal{R}$ have a fixed order. An entry (i, j) in the resulting matrix represents the longest path between the claim of resource r_j and the release of resource r_i (where r_j and r_i refer to the j-th and i-th element of $\mathcal{R}$ respectively). In the case a path does not exist the entry is set to $-\infty$. For the resources that are not used by the activity, all related entries to those resources are set to $-\infty$ with the exception of the entries (i, i) which are set to 0. This exception is necessary since even though the resource is not used, its availability time needs to be carried on to the updated resource time-stamp vector.

Example 3.2 Consider again activity *a* depicted in Figure 3.2 (a) and its (max,+) characterization:

$$\boldsymbol{M}_a = \begin{bmatrix} 2 & -\infty \\ 3 & 4 \end{bmatrix}$$

Note that each index (i, j) of the matrix corresponds to the longest path from the claim of resource r_j to the release of resource r_i. Figure 3.2 (b) depicts the longest paths for each pair of resources. For instance the longest path from (r_1, cl) to (r_1, rl) is equal to 2 time units thus $\boldsymbol{M}_a(1, 1) = 2$. No path exist between (r_2, cl) and (r_1, rl) and thus $\boldsymbol{M}_a(1, 2) = -\infty$.

Example 3.3 Consider the same activity as in Example 3.2, but now in a system with one additional resource r_3. In this case our system is composed from three resources, but our activity only requires two of them. The (max,+) matrix characterization of activity *a* must account for all the resources, not just the ones claimed by *a*. Since there are no paths from (and to) the non-utilized resources in activity *a* one could simply fill the corresponding entries with $-\infty$. However, doing so would imply the initial entry $\gamma_R(r_3)$ to be lost in the updated resource time-stamp vector $\boldsymbol{M}_a \otimes \gamma_R$. For this reason we keep the entry value corresponding to (r_3, cl) and (r_3, rl) as a fictitious dependency between (r_3, cl) and (r_3, rl) and hence set $\boldsymbol{M}_a(3, 3) = 0$. The corresponding (max,+) matrix of activity *a* therefore is:

$$\boldsymbol{M}_a = \begin{bmatrix} 2 & -\infty & -\infty \\ 3 & 4 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix}$$

3.2.3 Sequencing Activities using their (max,+) Characterization

Assuming an initial γ_R, the resource time-stamp vector after executing activity *a* can be computed as $update(\gamma_R, a) = \boldsymbol{M}_a \otimes \gamma_R$. If the execution of *a* is followed by an activity *b*, we can compute the new resource time-stamp vector as $update(update(\gamma_R, a), b) = update(\boldsymbol{M}_a \otimes \gamma_R, b) = \boldsymbol{M}_b \otimes \boldsymbol{M}_a \otimes \gamma_R$. In this fashion, the sequencing of activities *a*;*b* can be captured by repeated matrix multiplications, where each (max,+) matrix represents the temporal behavior of the corresponding activity. This is generalized in the following lemma:

Lemma 3.3 — ($(\max, +)$ sequencing characterization). Let $\underline{a} = a_1\ a_2\ ...\ a_n$ be an activity sequence and let γ_R be a resource time-stamp function. Then the updated resource time-stamp vector after the execution of $\underline{a}$ is given by:

$$update(\gamma_R, a_1; ...; a_n) = \boldsymbol{M}_{a_n} \otimes \boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_1} \otimes \gamma_R$$

Example 3.4 As an example consider the activities of Figures 3.2 (a) and 3.4 (a) which we will respectively refer to as *a* and *b*. The Gantt chart of the execution of the sequenced activity $a;b$ assuming starting resource time-stamp vector $\gamma_R = [2, 0]^T$ is depicted in Figure 3.4 (b). The updated resource time-stamp vector γ'_R after executing $a;b$ equals $update(\gamma_R, a; b)$ with $\gamma'_R = [10, 9]^T$ and is depicted in Figure 3.2 (b) by the small red circles with the resource number written within.
Now, consider the (max,+) matrices $\boldsymbol{M}_a$ and $\boldsymbol{M}_b$:

$$\boldsymbol{M}_a = \begin{bmatrix} 2 & -\infty \\ 3 & 4 \end{bmatrix} \quad \boldsymbol{M}_b = \begin{bmatrix} 6 & 5 \\ -\infty & 4 \end{bmatrix}$$

We can compute the updated vector after the execution $a; b$ following Lemma 3.3 where $update(\gamma_R, a; b)$ is equal to:

$$\begin{aligned} &= \boldsymbol{M}_b \otimes \boldsymbol{M}_a \otimes \gamma_R \\ &= \begin{bmatrix} max(6+2, 5+3) & max(6+-\infty, 5+4) \\ max(-\infty+2, 4+3) & max(-\infty+-\infty, 4+4) \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 8 & 9 \\ 7 & 8 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 10 \\ 9 \end{bmatrix} \end{aligned}$$

Note that indeed we obtain the same updated resource time-stamp vector $[10, 9]^T$.

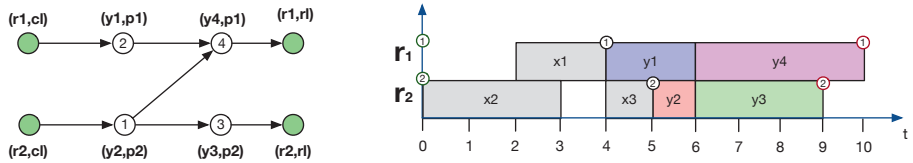


Figure 3.4: *(a) example activity* b*; (b) Gantt chart execution of activity* b *preceded by activity* a *(Figure 3.3) and (c) execution of activities* a *and* b *assuming the initial resource time-stamp vector* $\gamma_R = [2, 0]^T$.

3.3 (max,+) Automaton

In a logistics automaton each activity sequence represents a possible manufacturing of a batch of products where each activity represents a single manufacturing operation. The manufacturing behavior is captured by a language of activity sequences encoded by a logistics automaton. In the previous sections we have shown how the temporal behavior of a single activity and of the updated resource time-stamp vector due to its execution are described in (max,+) algebra. In this section we show how we capture the complete temporal behavior by a *(max,+) expansion* of the logistics automaton considering the (max,+) characterization of all specified 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 3.1 — ((max,+) automaton). Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a logistics automaton. First define MaxPlusStates(L) as the smallest set V satisfying inference rules (1) and (2):

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

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

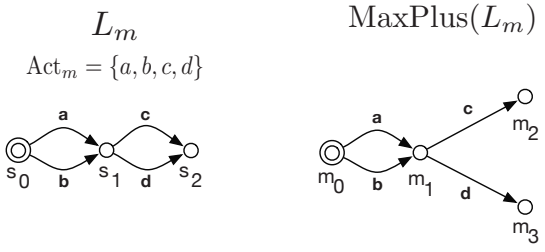
define MaxPlus(L) as

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

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

Note that a (max,+) automaton is a logistic automaton. 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. Nonetheless, the language of the (max,+) automaton is still equivalent to that of the initial logistics automaton.

Lemma 3.4 Given an logistics automaton L then $\mathcal{L}(L) = \mathcal{L}(\text{MaxPlus}(L))$.



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

Figure 3.5: *Example of the (max,+) expansion for logistics automaton L_m. M_a, M_b, M_c and M_D refer to the (max,+) matrices of activities a, b, c and d.*

Example 3.5 Consider the logistics automaton L_m and the (max,+) matrices of activities a, b, c and d depicted in Figure 3.5. The corresponding MaxPlus(L_m) of L_m is also depicted in Figure 3.5. Given that $m_0 = (s_0, \mathbf{0}_R)$ we have by Definition 3.1 that $m_1 = (s_1, [2, 2]^T)$, $m_2 = (s_2, [5, 6]^T)$ and

$m_3 = (s_2, [7, 5]^T)$. Note that due to the different $(max, +)$ matrices of activities c and d, state s_2 is duplicated in states m_2 and m_3 of MaxPlus(L_m). Vice-versa, since activities a and b have equal $(max, +)$ matrices, s_1 occurs only once in the states of MaxPlus(L_m). Finally notice that indeed the language of the (max,+) automata $\mathcal{L}(\text{MaxPlus}(L_m)) = \mathcal{L}(L_m)$, consistent with Lemma 3.4.

(max,+) Expansion of a Logistics Automaton

A (max,+) automaton can be constructed by transversing the original logistics automaton and computing for each visited state the resulting resource time-stamp vectors due to the execution of activities leading to that state (similarly to the approach used for computing worst-case throughput in SADF models [38]). The new state in the (max,+) automaton is a tuple composed by the original state in the logistics automaton and a resource time-stamp vector γ_R. Recall that for the initial states of the logistics automaton (states with no incoming edges) $\gamma_R = \mathbf{0}_R$. For instance, assume that two states s_1 and s_2 in the logistics automaton are such that $s_1 \xrightarrow{a} s_2$. These would then correspond to states $m_1 = (s_1, \gamma_R)$ and $m_2 = (s_2, \boldsymbol{M}_a \otimes \gamma_R)$, for some vector γ_R. Notice that whenever two sequences lead to the same state in a (max,+) automaton, the corresponding execution of the activities in the sequence leading to that state result in the same resource time-stamp vector. Algorithm 2 defines the necessary computation steps to obtain a (max,+) automaton from a logistics automaton. We use a depth-first search for this purpose. The while loop pops a (max,+) state from the stack to be the current state and marks it visited. For each possible transition from the current state, a new (max,+) state is created with a matching logistics automaton state and a resource time-stamp vector computed using the (max,+) matrix of the activity associated with that transition. If the new (max,+) state does not yet exist in the set of states of the (max,+) automaton then the new state and a transition from the current state to the new state are added to the (max,+) automaton. If the new state already exists, then only the transition is added. This while loop is repeated until the stack is empty.

3.3.1 Solving the BMO Problem

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*. A solution

Algorithm 2 Algorithm for constructing the (max,+) automaton

procedure CREATEMAXPLUSAUTOMATON($\mathcal{A}ct$, $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$)
  stackDSF = new empty stack;
  MaxPlus$(L) = \langle \mathcal{S}', \mathcal{A}ct, \dot{\rightarrow}', \mathcal{S}_0' \rangle$;
  $\mathcal{S}_0'$ = $\{(s_0, \mathbf{0}_R)\}$;
  add $(s_0, \mathbf{0}_R)$ to $\mathcal{S}'$;
  push $(s_0, \mathbf{0}_R)$ into stackDSF;
  while stackDSF not empty **do**
    $(s, \gamma) \leftarrow$ pop from stackDSF;
    nextTrans = $\{(s, a, s') \in \dot{\rightarrow} |\ a \in \mathcal{A}ct$ and $s' \in \mathcal{S}\}$;
    for each transition $s \xrightarrow{a} s'$ in nextTrans **do**
      $\gamma' \leftarrow \boldsymbol{M}_a \otimes \gamma$;
      if $(s', \gamma') \notin \mathcal{S}'$ **then**
        add (s', γ') to $\mathcal{S}'$;
        push (s', γ') to stackDSF;
      add $(s, \gamma) \xrightarrow{a} (s', \gamma')$ to $\dot{\rightarrow}'$;
  return MaxPlus(L)

to the BMO problem can be obtained by exploring 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. This proven in the following theorem.

Theorem 3.5 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 $(s, \gamma) \not\dot{\rightarrow}$ and that for all $(s', \gamma') \in \mathcal{S}$ with $(s', \gamma') \not\dot{\rightarrow}, \|\gamma\| \leq \|\gamma'\|$. Further let $\underline{a} \in \mathcal{L}(L)$ be such that $(s_0, \mathbf{0}_R) \xrightarrow{\underline{a}} (s, \gamma)$. Then for all $\underline{a}' \in \mathcal{L}(L)$:

$$mks(\underline{a}) \leq mks(\underline{a}')$$

Proof. Assume $(s, \gamma) \in \mathcal{S}$ is such that $(s, \gamma) \not\dot{\rightarrow}$ and that for all $(s', \gamma') \in \mathcal{S}$ with $(s', \gamma') \not\dot{\rightarrow}, \|\gamma\| \leq \|\gamma'\|$. Let $\underline{a} \in \mathcal{L}(L)$ be such that $(s_0, \mathbf{0}_R) \xrightarrow{\underline{a}} (s, \gamma)$. By Definition 2.7, $mks(\underline{a}) = \left\| update(\mathbf{0}_R, a_1; ...; a_n) \right\|$ and by Lemma 3.3 we thus have $mks(\underline{a}) = \left\| \boldsymbol{M}_{a_n} \otimes \cdots \otimes \boldsymbol{M}_{a_1} \otimes \mathbf{0}_R \right\|$. Since $(s_0, \mathbf{0}_R) \xrightarrow{\underline{a}} (s, \gamma)$ it follows

from Definition 3.1 that $\gamma = \boldsymbol{M}_{a_n} \otimes \cdots \otimes \boldsymbol{M}_{a_1} \otimes \mathbf{0}_R$. Thus $mks(\underline{a}) = \|\gamma\|$.

Now let $\underline{a}' \in \mathcal{L}(L)$. Then by Lemma 3.4 we have $\underline{a}' \in \mathcal{L}(\text{MaxPlus}(L))$. Therefore there exists a state (s', γ') such that $(s', \gamma') \not\rightarrow$ and $(s_0, \mathbf{0}_R) \xrightarrow{\underline{a}'} (s', \gamma')$. Therefore $mks(\underline{a}') = \|\gamma'\|$ and $\|\gamma\| \leq \|\gamma'\|$. But then $mks(\underline{a}) \leq mks(\underline{a}')$. ■

Example 3.6 Consider the optimization-space encoded by MaxPlus(L_m) depicted in Figure 3.5. Following Definition 3.1 we have $m_0 = (s_0, \mathbf{0}_R)$, $m_1 = (s_1, [2, 2]^T)$, $m_2 = (s_2, [5, 6]^T)$ and $m_3 = (s_2, [7, 5]^T)$. Note that m_2 and m_3 are final states with makespans $\left\|[5, 6]^T\right\| = 6$ and $\left\|[7, 5]^T\right\| = 7$, respectively. Thus the minimal makespan is 6 time units. Since activity sequences $a\,c$ and $b\,c$ both terminate in final state m_3 they are both solutions to the BMO problem induced by L_m.

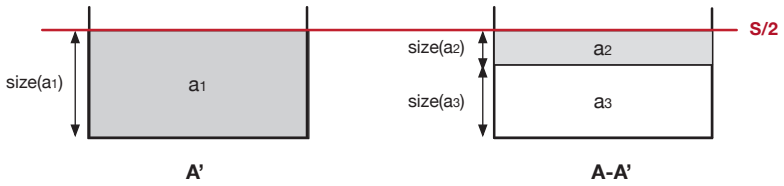


Figure 3.6: *Example to the solution of the WSPP given the set of of elements e_1, e_2 and e_3 of set A. Equal total sizes for each subset is obtained by placing element e_1 in subset A' and elements e_2 and e_3 in subset $A' \backslash A$.*

3.3.2 Complexity Analysis

The computational complexity of the BMO problem is shown to be *NP-Hard* in two steps. First we show that the decision version of the BMO problem, which we call the Batch Makespan Satisfaction (BMS) problem is *NP-Complete*. The BMS problem is concerned with the decision whether an activity sequence exists with a makespan lower than a given bound B. This is shown by reducing the Weighted Set Partitioning (WSP) problem [37], which is known to be NP-Complete [46], to BMS. Next we show the BMO problem to be NP-Hard by reducing BMS to BMO. Let us start by formally introducing the Weighted Set Partitioning (WSP) problem [37].

Problem 3.2 – (Weighted Set Partitioning problem). Assume a finite set $A = \{a_1, ..., a_n\}$ of n elements, where each element a_i with $1 \leq i \leq n$ has a postive size $size(a_i) \in \mathbb{Z}^+$. Does a subset $A' \subseteq A$ exist such that $\sum_{a \in A'} size(a) = \sum_{a \in A \setminus A'} size(a)$?

Example 3.7
As a simple example consider a set of elements $\{a_1, a_2, a_3\} \in A$ for which $size(a_1) = 4$, $size(a_2) = 1$ and $size(a_3) = 3$. Figure 3.6 depicts a solution to this WSP problem letting $A' = \{a_1\}$ and $A \setminus A' = \{a_2, a_3\}$.

Theorem 3.6 BMO is NP-hard.

Proof. We first reduce the WSP problem to our BMS problem in the following way. Let $A = \{a_1, ..., a_n\}$ and let $size(a_i)$ denote the size of element a_i ($1 \leq i \leq n$). We define two resources R_1 and R_2, where R_1 has peripheral p_1 and R_2 has peripheral p_2. For each i ($1 \leq i \leq n$) we define two activities a_i^1 and a_i^2:

- $a_i^1 : (R_1, cl) \rightarrow (x_i, p_1) \rightarrow (R_1, rl)$
- $a_i^2 : (R_2, cl) \rightarrow (x_i, p_2) \rightarrow (R_2, rl)$

Here x_i is an action referring to element a_i with execution time $T_{x_i} = size(a_i)$. We further define a logistics automaton with states $\mathcal{S} = \{s_1, ..., s_{n+1}\}$ and with transitions $s_i \xrightarrow{a_i^1} s_{i+1}$ and $s_i \xrightarrow{a_i^2} s_{i+1}$ ($1 \leq i \leq n$). It is not hard to see that a WSP partitioning exists if and only if the logistics automaton has an activity sequence exists with a makespan lower than or equal to $S/2$. Hence BMS is NP-Hard. It takes polynomial time to verify a possible solution to BMS, and thus BMS is NP-Complete. Now assume we have a solution $\underline{a_o}$ to BMO. If $mks(\underline{a_o})$ does not exceed a bound B, a solution with a lower makespan than B exists and therefore the answer to BMS is positive. Otherwise, no solution exists with makespan lower than B, yielding a negative answer to the BMS problem. Thus, BMO is NP-Hard.

■

3.4 Optimizing the Twilight

To illustrate the optimization steps we use our Twilight system running example. Consider logistics automaton $\big((L_1 \odot L_2) \restriction F_{in} \restriction F_{out}\big) \restriction C_{COND} \restriction$

$C_{DRILL} \restriction C_{LR} \restriction C_{UR}$ depicted in Figure 2.18 of Section 2.3.3.
The corresponding (max,+) automaton is obtained by transversing the logistics automaton of Figure 2.18 using Algorithm 2. The resulting automaton is shown in Figure 3.7. It represents the optimization-space of the two-product Twilight system and consists of 193 states and 273 transitions. As expected the (max,+) expansion results in a larger state-space than the state-space of the corresponding logistics automaton due to the branching effect when adding timing information. In the figure states are denoted as circles and transitions as directed edges. The states in the (max,+) automaton hold a state identifier from the corresponding state in the logistics automaton. In addition states contain a resource time-stamp vector capturing the resource availability times, but they are only shown in Figure 3.7 for final states. These final states are depicted in green boxes. The initial state is colored in cyan and the worst-case and best-case activity sequences are colored in red and green respectively. In total we obtained 14 distinct final states which implies that there can be 14 different temporal outcomes for the manufacturing of two products.

Table 3.1: *Makespan and resource availability times for each of the final states of the optimization-space.*

	Resource Availability Times					
Final State	*CA*	*COND*	*DRILL*	*LR*	*UR*	**Makespan**
1	65.6	26.9	63.6	55.9	69.4	69.4
2	61.1	39.2	59.1	51.4	64.9	64.9
3	65.3	43.8	63.3	28.9	69.1	69.1
4	68.1	46.2	66.1	58.4	71.9	71.9
5	67.7	46.2	65.7	41.8	71.5	71.5
6	66.3	44.4	64.3	56.5	70.1	70.1
7	70.5	49.0	68.5	36.0	74.3	74.3
8	67.1	40.4	65.1	57.4	70.9	70.9
9	68.5	46.6	66.5	58.8	72.3	72.3
10	68.1	46.6	66.1	42.2	71.9	71.9
11	64.8	42.9	62.8	55.1	68.6	68.3
12	69.0	47.5	67.0	22.5	72.8	72.8
13	60.0	33.3	58.0	50.3	63.8	63.8
14	58.6	30.0	56.6	48.9	62.4	62.4

Table 3.1 shows the makespan values and resource availability times for each

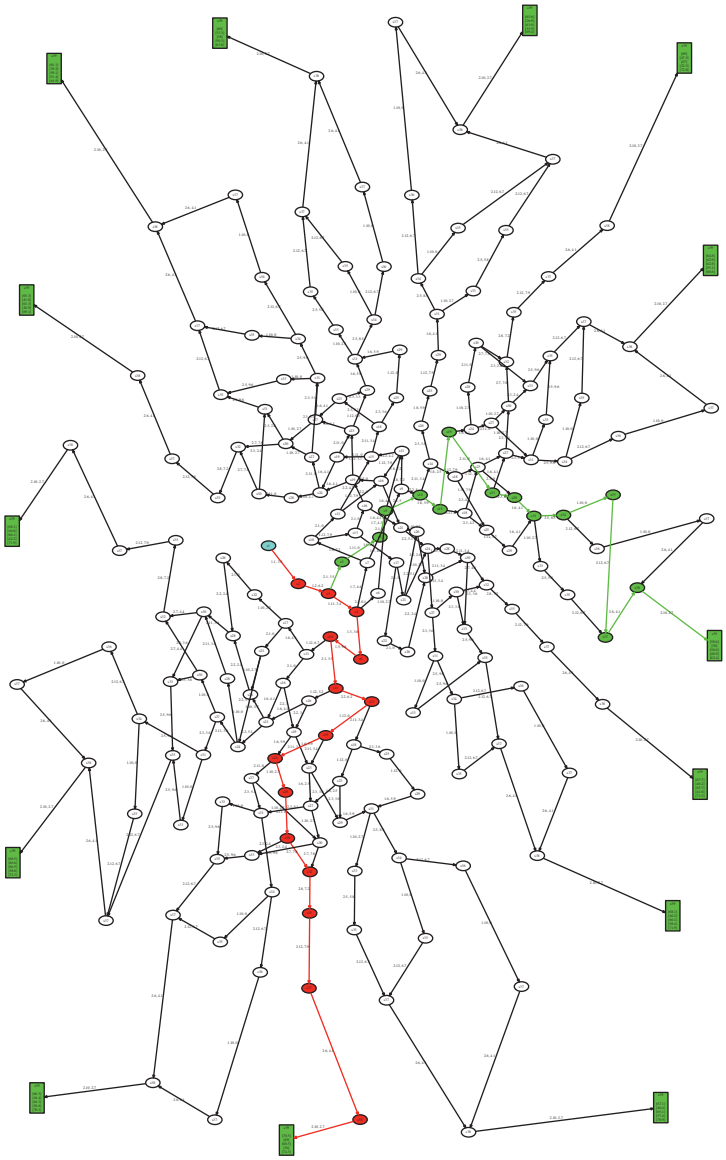


Figure 3.7: *Optimization-space of the Twilight system of Figure 2.18. Nodes filled in green represent final states and the node filled in cyan the initial state.*

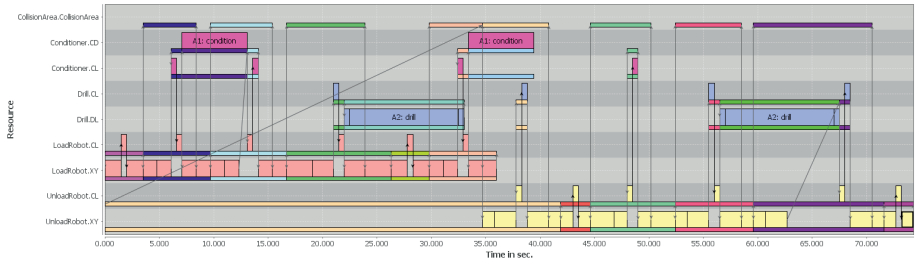


Figure 3.8: *Worst-case makespan activity sequence for the two-product Twilight system.*

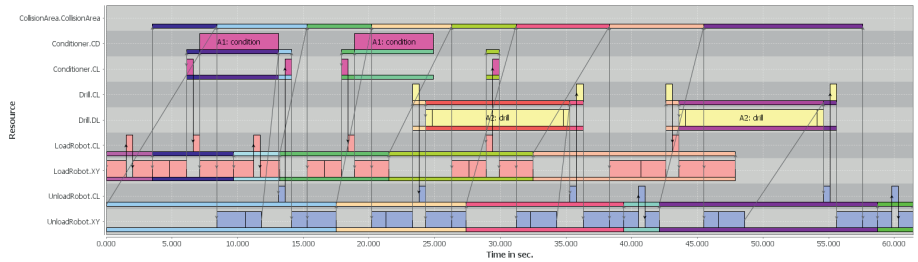


Figure 3.9: *Best-case makespan activity sequence for the two-product Twilight system.*

of the 14 final states. Observing the table we conclude that the range of the makespan for a two-product twilight system is [62.4,74.3] and that the optimal makespan of the systems is 62.4 time units and the worst-case makespan is 74.3 time units. Notice that resource availability times of the UR resource determine the makespan. This is because it is used as the last resource in each sequence. Notice also that in each final state the availability times of the remaining resources take different values. From the analysis we obtain the following worst-case ($\underline{wc}$) and best-case ($\underline{bc}$) activity sequences, which correspond to final states 7 and 14 respectively:

$$\underline{wc} = \mathbf{1}.1\ \mathbf{2}.1\ \mathbf{11}.1\ \mathbf{3}.1\ \mathbf{5}.1\ \mathbf{1}.2\ \mathbf{2}.2\ \mathbf{12}.1\ \mathbf{6}.1\ \mathbf{10}.1\ \mathbf{11}.2\ \mathbf{7}.2\ \mathbf{8}.2\ \mathbf{12}.2\ \mathbf{6}.2\ \mathbf{10}.2$$

$$\underline{bc} = \mathbf{1}.1\ \mathbf{2}.1\ \mathbf{1}.2\ \mathbf{11}.1\ \mathbf{7}.1\ \mathbf{2}.2\ \mathbf{8}.1\ \mathbf{12}.1\ \mathbf{11}.2\ \mathbf{3}.2\ \mathbf{6}.1\ \mathbf{5}.2\ \mathbf{10}.1\ \mathbf{12}.2\ \mathbf{6}.2\ \mathbf{10}.2$$

Figures 3.8 and 3.9 depict the Gantt charts of $\underline{wc}$ and $\underline{bc}$ respectively. The horizontal axis represents the action executed in time and the vertical axis

shows the horizontal lanes for each peripheral. Each horizontal lane of the Gantt chart depicts two types of boxes. Thick boxes represent the execution of an action by a peripheral and an action and a slim box represents the time the corresponding resource is claimed by an activity.

Notice that in the case of $\underline{wc}$ (Figure 3.8) the activity sequence corresponds to an almost complete sequential manufacturing of two products, where the system waits until one product is finished before starting to manufacture the next. The robot arms UR and LR are used sequentially and (with the exception of the input and output operations) that LR is entirely dedicated to product 1 and UR to product 2. On the other hand in sequence $\underline{bc}$ (Figure 3.9), the use of LR and UR is intertwined to exploit the pipelining of activities and thus resulting in a better performance. Moreover, we see that the claiming and releasing of the Collision Area (CA) is not well utilized in sequence $\underline{wc}$ since there are many gaps between accesses to the area in Figure 3.8. In the best-case sequence $\underline{bc}$ we see in Figure 3.9 that the CA resource is claimed uninterruptedly. This shows that the system is optimally utilizing the CA which might therefore constitute a bottleneck.

3.5 Related Work

In Chapter 2 we discussed the specification of flexible manufacturing systems with focus on functional aspects such as their capabilities and their logistics requirements. In this section, we complement this previous related work by considering the temporal aspects of flexible manufacturing systems. In particular, we focus on performance analysis and makespan optimization of batches of products in flexible manufacturing systems. In general, these performance analysis and optimization techniques are either based on simulations or on analytical techniques. We start by considering simulation-based approaches.

The CyPhySim tool [50], based on the Ptolemy II modeling language, and the Simulink toolset of MATLAB [57] provide a number of simulations models, such as ordinary differential equations, discrete/hybrid event models and discrete periodic systems, that enable system designers to study and analyze the performance of a flexible manufacturing system. For instance, in [65] a modeling approach in Simulink is proposed for the performance analysis of production systems. This includes the modeling and analysis of components such as robot arms and conveyor belts. Another example of a simulation-based

method is [45] in which a refinement-based design method using the POOSL [78] language is proposed. It covers multiple model abstractions including details on the timing information of the system which enable performance analysis. All these simulation approaches target the investigation such as the makespan of a batch. However, these simulations suffer from poor scalability and long simulation times and are therefore not well-suited for design exploration targeted in this thesis.

With respect to analytical techniques we can find many related approach in the field of job-shop scheduling. In [6], [7] a survey of several approaches using job shop scheduling to solve different instances of manufacturing systems. In general, job shop related problems do not consider resource sharing, multiple resource assignments or different possible routings, which are important aspects of flexible manufacturing systems addressed in this thesis. Most closely related work in this field addressing the design exploration of flexible manufacturing systems is [82]. Here a systematic methodology is proposed to explore different system configurations (i.e. number of resources, operation assignment or shared areas). For each selected configuration, a makespan minimization problem is formulated in terms of a job shop scheduling problem. The framework then relies on specific analysis techniques to find a solution to that particular problem. With respect to functional requirements these are posteriorly verified by model-checking. In contrast, our approach is general and can optimize any problem that can be specified in our framework. Further functional requirements are enforced by construction. We will discuss further on heuristics and on the scalability of our approach in Chapter 4.

Makespan optimization is also address in the Petri Net domain. In [69] a Petri Net model of a Multi-Robot System (MRS) is made where several robots operate simultaneously on the same product. The goal is the design exploration of different configurations (i.e. number of resources and operation assignment). The system in question exhibits many of the aspects of flexible manufacturing systems such as resource sharing and multiple resource assignments. Resource sharing and access to mutually exclusive resources (i.e. share areas/collision areas) is taken into account implicitly by conservative execution time estimates for the movement of each robot resource. Functional requirements are not explicitly taken into account. In contrast, our approach takes explicitly resource sharing into account as well as functional requirements, either in the design of the activities or in the formulation of logistics requirements. In [48], [51], [61]

Petri Net models are used for scheduling optimization of batches of products. Here the solutions are not guaranteed to be optimal and functional properties like safety are ignored. On the contrary, our approach guarantees optimal solutions with respect to the specified functional requirements.

3.6 Conclusions

In this chapter we introduced the necessary concepts and methods for the optimization domain of our framework such that different design specifications can be evaluated in terms of their expected performance. Since we focus on flexible manufacturing systems which often work with small product batches and a mix of different product types we introduced and defined the Batch Makespan Optimization (BMO) problem. We provided a complexity analysis and showed that the BMO falls within the class of NP-Hard problems. A solution to the BMO problem can be obtained by finding the activity sequence with the lowest makespan within the language of activity sequences of the specified logistics automaton. In order to efficiently compute the makespan of an activity sequence, we introduced (max,+) algebra semantics for activities. The temporal behavior of each activity is captured by a single (max,+) matrix and an initial resource time-stamp vector. The makespan of an activity sequence is then efficiently computed by a series of (max,+) matrix multiplications. A (max,+) automaton was introduced for which each activity sequence represents the correct manufacturing of a batch of products for which the completion time is found by computing the norm of the resource time-stamp vector of its final state. We showed that the (max,+) automaton is obtained by the means of a (max,+) expansion of a logistics automaton with the (max,+) characterization of all system activities and that a solution for the BMO problem can be obtained by finding an activity sequence in the optimization-space terminating in a final state with the lowest resource time-stamp vector norm. Finally we use the two-product Twilight system example to show how these concepts and methods can be used in practice to compute best and worst-case makespan values for a manufacturing system.

4 | Exploiting Constraints to Reduce the Optimization-space

Chapters 2 and 3 introduced the *specification* and *optimization* steps of our design exploration and optimization framework for flexible manufacturing systems. The framework is able to determine the optimal makespan activity sequence for a batch of products of a given system specification. The modularity of the framework allows for the specification of complex manufacturing systems and their requirements. However, as it was shown in Chapter 3 the Batch Makespan Optimization (BMO) problem induced by the specified logistics automaton, falls under the class of NP-Hard problems. As a consequence, optimal solutions might take prohibitively long depending on the size of the optimization-space induced by the optimization problem. To cope with this complexity, in this chapter we will develop an algebra of logistics automata to reason in a modular (algebraic) way about (behavioural and structural) *equivalence* and *inclusion* relations between logistics automata. These allow us to systematically relate their languages, their state-space and optimization-space sizes and their solutions to the BMO problem. We will prove that these relations are substitutive under the MaxPlus, $\odot$, $\restriction$ and Tree (which we introduce in this chapter) operators and discuss the commutativity, associativity and distributivity of the operators.

To support optimization, we could develop heuristic approaches. These however, typically come at the cost of sub-optimal solutions and uncertainty regarding their quality. In this chapter we introduce a novel approach where we exploit the modular constraints of the framework as an alternative to heuristic solutions which 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 [74] and in which over-specification is used implicitly and unconsciously to deal with complexity. Examples of over-specification that we have encountered in industrial cases are disallowing multiple mapping possibilities for an operation or enforcing the static ordering of system operations. Our approach allows system designers to deal consciously with over-specification by an explicit formalization in terms of constraints. Counter intuitively, in general the constraining of a logistics automaton neither leads to state-space reduction nor to optimization-space reduction. Therefore in this chapter we establish sufficient conditions on logistics automata and constraints that do lead to these reductions. To this end we extend our algebra with the notions of *nonpermutation-repulsiveness* and *permutation-attractiveness* and prove that the constraining of a nonpermutation-repulsing logistics automaton with a permutation-attracting constraint always results in a reduced state-space and optimization-space.

The chapter is organized as follows. In Section 4.1 we discuss the growth of the optimization-space through an example of a buffered variant of the Twilight system. In Section 4.2 we define equivalence and inclusion relations on logistics automata. In Section 4.3 we introduce the concept of Tree automata to capture the worst-case optimization-space in terms of state-space size. In Section 4.4 we discuss the substitutivity of the inclusion and equivalence relations under all operators and the commutativity, associativity and distributivity properties of the composition and constraining operators. Section 4.5 establishes and proves the conditions under which the constraining of a logistics automaton leads to state-space and optimization-space reduction. For this purpose we introduce nonpermutation-repulsing and permutation-attracting automata. In Section 4.6 we show how the algebra is used to systematically relate logistics automata regarding their languages, their state-space and optimization-space sizes and their BMO solutions. In Section 4.7 we exemplify the application of the algebra by using over-specification as a means to reduce the optimization-space of the buffered variant of the Twilight system. Finally, Section 4.8 discusses related work and Section 4.9 concludes the chapter.

4.1 Growth of the Optimization-space

In Chapter 3 we discussed how to compute the *optimization-space* of a system specification via the (max,+) expansion of the logistics automaton encoding

the language of allowed activity sequences. This optimization-space is then used to compute the makespan optimal activity sequence by searching for a final state of the optimization-space which minimizes the norm of its resource time-stamp vector and by computing an activity sequence leading to that final state.

During (max,+) expansion the state-space of the logistics automaton can grow and in some cases this growth can be exponential (see Section 3.3.2). The growth of the optimization-space is due to the multiplication of states of the logistics automaton in the (max,+) automaton during the expansion step. Each state of the logistics automaton can occur multiple times in the (max,+) automaton depending on the cumulative products of (max,+) matrices along the paths from the initial state to this particular state. In the worst case the (max,+) expansion generates a (max,+) automaton where each state has a single incoming transition (this (max,+) automaton is then a Tree automaton as will be discussed in Section 4.3). But also the nature of the system specification plays a role. For instance, the *flexibility* exhibited by a system (e.g. assignment and routing choices) and its requirements (e.g. input/output orders and product flow) have an impact on the state-space of of the logistics automaton which directly influences the size of the (max,+) automaton.

4.1.1 Buffered Twilight

To illustrate the growth of the optimization-space leading to infeasible optimization, we introduce a variation of the Twilight which we call *Buffered Twilight*. Later in this section we will revisit this example and use constraints to effectively reduce the optimization-space and compute bounds on the optimal solution. In this variant, a *Buffer* resource of capacity two is added to the resources of the Twilight system. At each step in the product flow (i.e. after the input, conditioning or drilling of a product) the system is allowed to place/pick a product on the Buffer resource, thus allowing the out-of-order execution of different products, without relaxing the end-to-end FIFO requirements. For example, after product i is inputted it can be placed on the buffer after which the following product $i+1$ can be inputted and placed on the COND resource for conditioning. In this way product $i+1$ executes activity Condition before product i, which is not possible in the regular Twilight system. We assume that placing a product on the Buffer resource after conditioning does not alter its conditioning properties (i.e. the product remains aligned and at a correct temperature).

Table 4.1: *Size of the state-space of the logistics automaton and optimization-space of the regular Twilight system for different batch sizes.*

	State-Space		Optimization-Space	
Batch Size	N. States	N. Edges	N. States	N. Edges
2	43	65	170	243
4	178	484	1013	1820
6	323	690	1949	3588
8	487	9760	2885	5365
10	647	1229	3821	7124

Table 4.2: *Size of the state-space logistics automaton and optimization-space of the Buffered Twilight variant for different batch sizes.*

	State-Space		Optimization-Space	
Batch Size	N. States	N. Edges	N. States	N. Edges
2	174	353	5227	8528
4	5700	13872	2344026	3960469
6	23464	57093	-	-
8	-	-	-	-
10	-	-	-	-

To take advantage of the added Buffer resource, the Buffered Twilight permits the execution of mixed product types. In this variant, we assume that the system is able to perform two distinct product flows: the *regular* Twilight product flow (as discussed in Section 2.1.3) and a *double pass* product flow (where a product passes twice on the Drill processing unit for a more detailed profiling). Further, we assume that the batch to be processed consists of an alternating sequence of *regular* and *double-pass* products. When we continue with this example in Section 4.7 we will give a detailed specification.

Table 4.1 and Table 4.2 show the size of the state-space of the logistics automaton and of the optimization-space for batches of 2, 4, 6, 8 and 10 products for the regular Twilight system and for the Buffered Twilight systems. We denote state-spaces and optimization-spaces that could not be computed by placing

a dash (-) in the respective N. States and N. Edges. Notice that in case of the regular Twilight system the optimization-space seems to grow in linear pace, while in case of the Buffered Twilight it seems to grow much faster. From a batch of 2 products to a batch of 4 products the number of states jumps from 5227 to 3960469. Moreover, we are no longer able to compute the optimization-space from 6 products onwards. This exemplifies the observation that in some cases we might encounter an exponential growth of the optimization-space. In Section 4.7 we will explain in detail the specification of the Buffered Twilight and use the algebra introduced in this chapter to prune the optimization-space so that (sub-optimal) solutions for larger batch sizes can be found. For this purpose we exploit over-specification by further constraining the logistics automata of the Buffered Twilight with additional non-essential constraints formalized as constraint automata.

4.2 Equivalence and Inclusion

In this section we define 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. We start by defining a strong equivalence relation on logistics automata.

Definition 4.1 — (Equivalence). Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ be logistics automata. Then L_1 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 $\mathcal{F} : \mathcal{S}_1 \rightarrow \mathcal{S}_2$ satisfying:

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

It is easy to show that $\approx$ is reflexive, transitive and symmetric. Hence:

Theorem 4.1 $\approx$ is an equivalence relation on logistics automata.

Equivalent logistics automata encode the same languages, the proof of which easily follows from Definitions 2.10 and 4.1.

Lemma 4.2 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 4.2 — (Inclusion). Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ be logistics automata. Then L_1 is included in L_2, written $L_1 \sqsubseteq L_2$, if and only if $\mathcal{A}ct_1 = \mathcal{A}ct_2$ and either i) $\mathcal{S}_1 = \emptyset$ or ii) $\mathcal{S}_{0_1} = \{s_{0_1}\}$ and $\mathcal{S}_{0_2} = \{s_{0_2}\}$ and there exists a relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ which is injective (i.e. for all $(s_1, s_2), (s_1', s_2') \in R$, $s_2 = s_2'$ implies $s_1 = s_1'$) and satisfies the following conditions:

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

where injectivity

Theorem 4.3 $\sqsubseteq$ 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 $L_2 \sqsubseteq L_1$.

Proof. We will first shown that $\sqsubseteq$ is reflexive, transitive and anti-symmetric.

Reflexivity: Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$. If $\mathcal{S} = \emptyset$ then by definition $L \sqsubseteq L$. Otherwise $\mathcal{S}_0 = \{s_0\}$. In that case define $R = \{(s, s) \mid s \in \mathcal{S}\}$. R satisfies conditions *1., 2. and 3.* of Definition 4.2 and hence also in this case $L \sqsubseteq L$.

Transitivity: Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$, $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ and $L_3 = \langle \mathcal{S}_3, \mathcal{A}ct_3 \dot{\rightarrow}_3, \mathcal{S}_{0_3} \rangle$ be logistics automata such that $L_1 \sqsubseteq L_2$ and $L_2 \sqsubseteq L_3$. Then clearly $\mathcal{A}ct_1 = \mathcal{A}ct_3$. Further either i) $\mathcal{S}_1 = \emptyset$ or $\mathcal{S}_2 = \emptyset$ or ii) $\mathcal{S}_{0_1} = \{s_{0_1}\}$, $\mathcal{S}_{0_2} = \{s_{0_2}\}$ and $\mathcal{S}_{0_3} = \{s_{0_3}\}$ for some $s_{0_1} \in \mathcal{S}_1$, $s_{0_2} \in \mathcal{S}_2$ and $s_{0_3} \in \mathcal{S}_3$. In case i) $\mathcal{S}_1 = \emptyset$ and thus $L_1 \sqsubseteq L_3$ by definition. In case ii) there exist relations $R_1 \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ and $R_2 \subseteq \mathcal{S}_2 \times \mathcal{S}_3$ that both satisfy conditions *1., 2. and 3.* of Definition 4.2. It is easy to verify that $R_2 \circ R_1$ also satisfies these conditions, where composition $R_2 \circ R_1$ is defined by $\{(s_1, s_3) \in \mathcal{S}_1 \times \mathcal{S}_3 \mid (s_1, s_2) \in R_1$ and $(s_2, s_3) \in R_2$ for some $s_2 \in \mathcal{S}_2\}$.

Anti-symmetry: Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$

be logistics automata with $L_1 \sqsubseteq L_2$ and $L_2 \sqsubseteq L_1$. We have to show that $L_1 \approx L_2$. Now clearly $\mathcal{A}ct_1 = \mathcal{A}ct_2$. Further either i) $\mathcal{S}_{0_1} = \mathcal{S}_{0_2} = \emptyset$ or ii) $\mathcal{S}_{0_1} = \{s_{0_1}\}$ and $\mathcal{S}_{0_2} = \{s_{0_2}\}$ for some $s_{0_1} \in \mathcal{S}_1$ and $s_{0_2} \in \mathcal{S}_2$. In case i) the result follows directly from Definition 4.1. In case ii) relations $R_1 \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ and $R_2 \subseteq \mathcal{S}_2 \times \mathcal{S}_1$ exist, both satisfying conditions *1., 2. and 3.* of Definition 4.2. Since R_1 is injective $\#\mathcal{S}_1 \leq \#\mathcal{S}_2$ (where we let $\#\mathcal{S}$ denotes the number of states in $\mathcal{S}$). Likewise $\#\mathcal{S}_2 \leq \#\mathcal{S}_1$ and thus $\#\mathcal{S}_1 = \#\mathcal{S}_2$. But then both R_1 and R_2 must be bijective functions. We claim that R_1 satisfies conditions *1. and 2.* of Definition 4.1. Condition *1.* is obviously true because R_1 satisfies condition *1.* of Definition 4.2. (Note that since R_1 is a bijection we can write $R_1(s_{0_1}) = s_{0_2}$ as an alternative to $(s_{0_1}, s_{0_2}) \in R_1$). To establish condition *2* (of Definition 4.1) let $s, s' \in \mathcal{S}_1$ and $a \in \mathcal{A}ct$. We have to show that $s \xrightarrow{a} s'$ if and only if $R_1(s) \xrightarrow{a} R_1(s')$. The "only if" part readily follows from condition *2.* of Definition 4.2. For the "if" part notice that $R_1(s) \xrightarrow{a} R_1(s')$ implies that $R_2(R_1(s)) \xrightarrow{a} R_2(R_1(s'))$ which implies that $R_1(R_2(R_1(s))) \xrightarrow{a} R_1(R_2(R_1(s')))$ and $R_2(R_1(R_2(R_1(s)))) \xrightarrow{a} R_2(R_1(R_2(R_1(s'))))$ etcetera. Hence for any $n \geq 1$ $(R_2 \circ R_1)^n(s) \xrightarrow{a} (R_2 \circ R_1)^n(s')$. Now $R_2 \circ R_1$ is an element of the finite group of permutations on $\mathcal{S}_1$. So when we compute $(R_2 \circ R_1)^1, (R_2 \circ R_1)^2, \cdots$ we must eventually get $(R_2 \circ R_1)^i = (R_2 \circ R_1)^j$ for some $j > i$. But then $(R_2 \circ R_1)^{j-i}$ is the identify permutation I (for which $I(s) = s$ for each $s \in \mathcal{S}_1$). Filling in $j - i$ for n yields $I(s) \xrightarrow{a} I(s')$ and therefore $s \xrightarrow{a} s'$.

Finally to show that $L_1 \approx L_2$ if and only if $L_1 \sqsubseteq L_2$ and $L_2 \sqsubseteq L_1$ we first observe that the "if" part immediately follows from the proof of anti-symmetry. For the "only if" part we claim that the bijective function (required in Definition 4.1) also satisfies conditions *1., 2. and 3.* of Definition 4.2. ■

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, which follows directly from the proof of anti-symmetry of the inclusion relation.

Lemma 4.4 Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ be logistics automata such that $L_1 \sqsubseteq L_2$. Then $\#\mathcal{S}_1 \leq \#\mathcal{S}_2$.

Automata inclusion carries over to language inclusion, which is stated in the following lemma.

Lemma 4.5 Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, \mathcal{S}_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, \mathcal{S}_{0_2} \rangle$ be logistics automata such that $L_1 \sqsubseteq L_2$. Then $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$.

Proof. If $\mathcal{S}_1 = \emptyset$ then the claim holds vacuously. Otherwise $\mathcal{S}_{0_1} = \{s_{0_1}\}$ and $\mathcal{S}_{0_2} = \{s_{0_2}\}$. We then have to show that for all $\underline{a} \in \mathcal{L}(L_1)$, $\underline{a} \in \mathcal{L}(L_2)$. If $\underline{a} = a_1 \cdots a_n \in \mathcal{L}(L_1)$ then for some $s_1, \cdots, s_n \in \mathcal{S}_1$, $s_{0_1} \xrightarrow{a_1} s_1 \cdots \xrightarrow{a_n} s_n$ and $s_n \not\rightarrow$. Since $L_1 \sqsubseteq L_2$ there exists a relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ satisfying the conditions of Definition 4.2. Therefore for some $s'_1, \cdots, s'_n \in \mathcal{S}_2$, $s_{0_2} \xrightarrow{a_1} s'_1 \cdots \xrightarrow{a_n} s'_n$ and $s'_n \not\rightarrow$. Hence $\underline{a} \in \mathcal{L}(L_2)$. Notice that condition *3.* of Definition 4.2 is essential for lemma to be true. In fact, this lemma is the key reason why this condition was posed in the first place.

■

Example 4.1 Consider logistics automata L_f and L_g depicted in Figure 4.1. These automata have the same language $\{a, b\}$. It is easy to check that L_f is included in L_g but that L_g is not included in L_f. Thus, even though L_f and L_g have the same language they are not equivalent.

Example 4.2 Consider logistics automata L_h and L_i depicted in Figure 4.2. 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 therefore not equivalent, even though they have the same language. Note that our inclusion relation is stronger than Milner's [59] 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 $L_i \not\sim L_h$, where $\sim$ denotes Milner's strong bisimulation. The essential property that makes $\sqsubseteq$ into a partial order is the property of injectivity.

Example 4.3 Consider logistics automata L_j and L_k depicted in Figure 4.3. Even though $L_j \prec L_k$ (where $\prec$ denotes Milner's simulation relation), $L_j \not\sqsubseteq L_k$. This is because the inclusion relation enforces (through condition *3.* of Definition 4.2) all activity sequences of $\mathcal{L}(L_j)$ to be included in $\mathcal{L}(L_k)$.

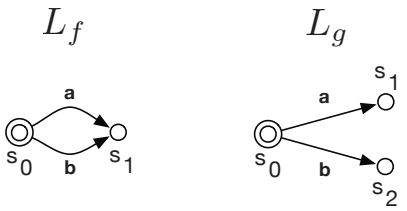


Figure 4.1: *Example logistics automata L_f and L_g. L_f is included in L_g, but L_f and L_g are not equivalent.*

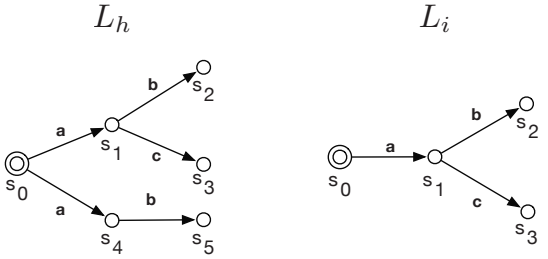


Figure 4.2: *Example logistics automata L_h and L_i. L_i is included in L_h but L_h and L_i are not equivalent.*

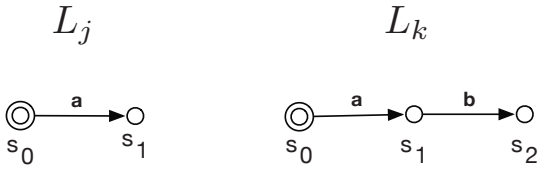


Figure 4.3: *Example where logistics automaton L_j is not included in logistics automaton L_k.*

4.3 Worst-case Optimization-space

In this section we introduce Tree automata to capture the worst-case encoding of the optimization-space and show that for any logistics automaton L, $L \sqsubseteq$ MaxPlus$(L) \sqsubseteq$ Tree(L). We start by looking at the inclusion relation between a logistics automaton and its corresponding (max,+) automaton.

Recall that each state of a logistics automaton can occur multiple times in the (max,+) automaton, depending on the cumulative products of (max,+) 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. Even stronger, the logistics automaton is included in the (max,+) automaton as stated by the following lemma.

Lemma 4.6 Let L be a logistics automaton. Then $L \sqsubseteq$ MaxPlus(L).

Proof. In case L has an empty set of states, the result holds vacuously. Otherwise the proof is established by defining a relation relating each state s in L to every state of the form (s, γ_R) in MaxPlus(L). It is easy to verify that this relation is injective and satisfies conditions *1., 2. and 3.* of Definition 4.2. ■

Example 4.4 Consider the logistics automaton L_m and the (max,+) matrices of activities a, b, c and d depicted in Figure 4.4. The corresponding MaxPlus(L_m) of L_m is also depicted in Figure 4.4. Given that $m_0 = (s_0, \mathbf{0}_R)$ we have $m_1 = (s_1, [2, 2])$, $m_2 = (s_2, [5, 6])$ and $m_3 = (s_2, [7, 5])$ according to Definition 3.1. Notice that L_m is included in MaxPlus(L_m). Note also that due to the different $(max, +)$ matrices of activities c and d, state s_2 is duplicated in states m_2 and m_3 of MaxPlus(L_m). On the other hand, since activities a and b have equal $(max, +)$ matrices s_1 occurs only once in the states of MaxPlus(L_m).

As discussed in Section 4.1, the size of the state-space of a (max,+) automaton can grow exponentially in the size of the state-space of the corresponding logistics automaton. It is exactly this reason that makes the makespan optimization problem 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 and we will show that indeed MaxPlus$(L) \sqsubseteq$ Tree(L).

Definition 4.3 — (Tree automaton). Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, \mathcal{S}_0 \rangle$ be a logistics automaton. We first define Paths(L) as the smallest set V satisfying:

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

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

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

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

Lemma 4.7 Let $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, S_0 \rangle$ be a logistic automaton. Then MaxPlus$(L) \sqsubseteq$ Tree(L).

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

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

It is not hard to show that R satisfies conditions *1., 2. and 3.* of Definition 4.2. To prove that R is injective we have to show that any two pairs in R with equal right-hand elements also have equal left-hand elements. We will do this by induction on the depth of the derivation tree of one of the pairs. Notice that the pairs are either both derived by inference rule (1) or both derived by inference rule (2). In the first case their left-hand elements are obviously the

same. In the other case the pairs must be of the forms $((s', M_a \otimes \gamma_R), q\ s\ a\ s')$ and $((s', M_a \otimes \gamma'_R), q\ s\ a\ s')$ and we further know that $((s, \gamma_R), q\ s) \in R$ and $((s, \gamma'_R), q\ s) \in R$. By induction we then have $\gamma_R = \gamma'_R$. ■

Even though for each logistic automaton L we have $L \sqsubseteq \text{MaxPlus}(L) \sqsubseteq \text{Tree}(L)$, we claim without proof that these automata encode precisely the same language.

Lemma 4.8 Let L be a logistic automaton. Then $\mathcal{L}(L) = \mathcal{L}(\text{MaxPlus}(L)) = \mathcal{L}(\text{Tree}(L))$.

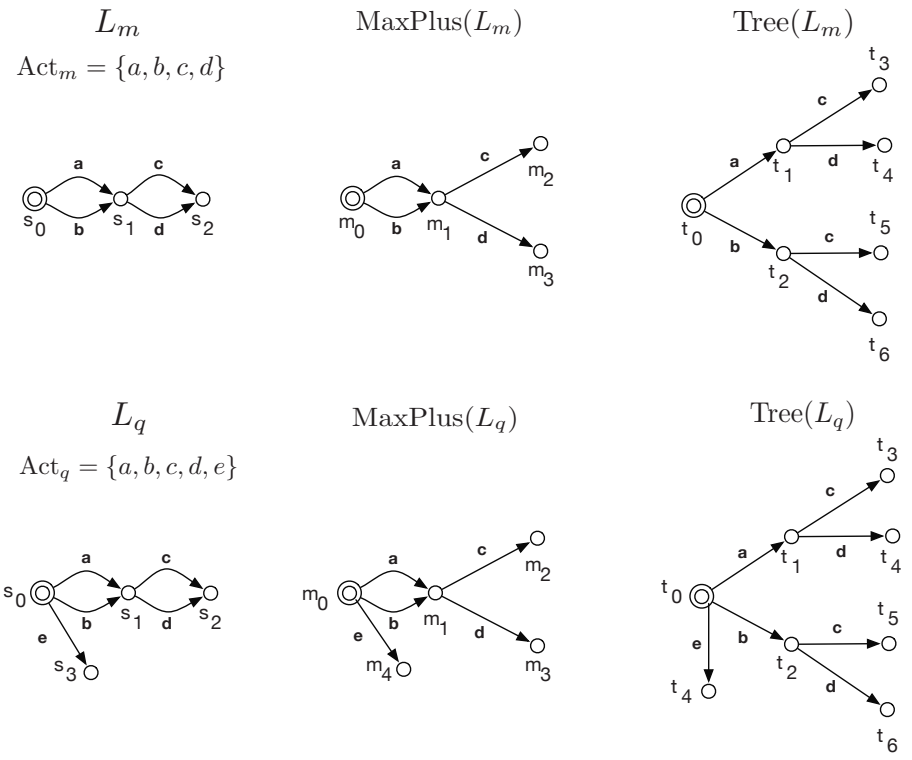
Example 4.5 Consider logistics automaton L_m depicted in Figure 4.4 again, together with its (max,+) and Tree counterparts. It is easy to see that both L_m and MaxPlus(L_m) are included in Tree(L_m). Note that each state in Tree(L_m) has a single incoming transition (with the exception of the initial state). Therefore the tree automaton has the shape of a tree. Further notice that each state in the tree automaton represents a unique path in the original automaton. For instance $t_6 = s_0\ b\ s_1\ d\ s_2$ represent that L_m can move from s_0 to s_2 via activities b and d and intermediate state s_1. Finally observe that the automata encode the same language $\{ac, ad, bc, bd\}$ and note their increasing number of states.

A property that will appear to be very useful is that our equivalence and inclusion relations are substitutive under both the MaxPlus and Tree operators:

Lemma 4.9 — (Substitutivity 1). Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ be logistics automata. Then $L_1 \sqsubseteq L_2$ implies MaxPlus(L_1) $\sqsubseteq$ MaxPlus (L_2) and Tree$(L_1) \sqsubseteq$ Tree(L_2). Further $L_1 \approx L_2$ implies MaxPlus$(L_1) \approx$ MaxPlus(L_2) and Tree$(L_1) \approx$ Tree(L_2).

Proof. Substitutivity of $\approx$ follows directly from the substitutivity of $\sqsubseteq$. We first prove substitutivity for the MaxPlus operator and then for the Tree operator.

MaxPlus: if $\mathcal{S}_1 = \emptyset$ the result follows immediately. Otherwise $S_{0_1} = \{s_{0_1}\}$ and $S_{0_2} = \{s_{0_2}\}$ and an injective relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ exists that satisfies conditions *1., 2.* and *3.* of Definition 4.2. Based on R we define a new relation $R' \subseteq$ MaxPlusStates$(L_1) \times$ MaxPlusStates(L_2) as the smallest set V that satisfies the following inference rules:



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

Figure 4.4: *Example where logistics automata L_m and L_q are included in their $(max, +)$ and $tree$ counterparts. M_a, M_b, M_c, M_d and M_e refer to the (max,+) matrices of activities a, b, c, d and e.*

$$\frac{\mathcal{S}_{0_1} = \{s_{0_1}\} \quad \mathcal{S}_{0_2} = \{s_{0_2}\}}{((s_{0_1}, \mathbf{0}_R), (s_{0_2}, \mathbf{0}_R)) \in V} \ (1)$$

$$\frac{((s_1, \gamma_R), (s_2, \gamma_R)) \in V \quad s_1 \xrightarrow{a}_1 s_1' \quad s_2 \xrightarrow{a}_2 s_2' \quad (s_1', s_2') \in R}{((s_1', M_a \otimes \gamma_R), (s_2', M_a \otimes \gamma_R)) \in V} \ (2)$$

It is easy to show that R' is injective and satisfies conditions *1.*, *2.* and *3.* of Definition 4.2.

Tree: if $\mathcal{S}_1 = \emptyset$ the result follows immediately. Otherwise $S_{0_1} = \{s_{0_1}\}$ and $S_{0_2} = \{s_{0_2}\}$ and an injective relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ exists that satisfies conditions *1.*, *2.* and *3.* of Definition 4.2. Based on R we define a new relation $R' \subseteq Paths(L_1) \times Paths(L_2)$ as the smallest set V that satisfies the following inference rules:

$$\frac{\mathcal{S}_{0_1} = \{s_{0_1}\} \quad \mathcal{S}_{0_2} = \{s_{0_2}\}}{(s_{0_1}, s_{0_2}) \in V} \ (1)$$

$$\frac{(q_1\ s_1, q_2\ s_2) \in V \quad s_1 \xrightarrow{a}_1 s_1' \quad s_2 \xrightarrow{a}_2 s_2' \quad (s_1', s_2') \in R}{(q_1\ s_1\ a\ s_1', q_2\ s_2\ a\ s_2') \in V} \ (2)$$

Here $s_1, s_1' \in \mathcal{S}_1$ and $s_2, s_2' \in \mathcal{S}_2$ and $a \in \mathcal{A}ct_1$. It is easy to show that R' satisfies conditions *1.*, *2.* and *3.* of Definition 4.2. To show that R' is injective we show that any two pairs in R' with equal right-hand elements also have equal left-hand elements. To this end notice first that the pairs are either both derived from rule (1) or both from rule (2). If the pairs are derived by inference rule (1) then the left elements are obviously the same. If the pairs are derived by inference rule (2) then they are of the forms $(q_1\ s_1\ a\ s_1', q_2\ s_2\ a\ s_2')$ and $(q_1'\ s_1\ a\ s_1', q_2\ s_2\ a\ s_2')$. This implies that $(q_1\ s_1, q_2\ s_2) \in R'$ and $(q_1'\ s_1, q_2\ s_2) \in R'$ and by induction we then know that $q_1 = q_2$. Thus R' is injective.

■

Substitutivity also holds for language equivalence, which follows directly from Lemma 4.8.

Lemma 4.10 — (Substitutivity 2). Let L_1 and L_2 be logistics automata. Then $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ implies $\mathcal{L}(\text{MaxPlus}(L_1)) \subseteq \mathcal{L}(\text{MaxPlus}(L_2))$ and $\mathcal{L}(\text{Tree}(L_1)) \subseteq \mathcal{L}(\text{Tree}(L_2))$.

Example 4.6 Consider the logistics automata L_m and L_q and the (max,+) matrices of activities a, b, c, d and e depicted in Figure 4.4. The corresponding MaxPlus(L_m), MaxPlus(L_q), Tree(L_m) and Tree(L_q) are also depicted in Figure 4.4. First notice that $L_m \sqsubseteq L_q$. As stated in Lemma 4.9, MaxPlus(L_m) $\sqsubseteq$ MaxPlus(L_q) and Tree(L_m) $\sqsubseteq$ Tree(L_q). Note that $\mathcal{L}(L_m) = \mathcal{L}(\text{MaxPlus}(L_m)) = \mathcal{L}(\text{Tree}(L_m)) = \{ac, ad, bc, bd\}$ and that $\mathcal{L}(L_q) = \mathcal{L}(\text{MaxPlus}(L_q)) = \mathcal{L}(\text{Tree}(L_q)) = \{ac, ad, bc, bd, e\}$ (is consistent with Lemma 4.8) and also $\mathcal{L}(\text{MaxPlus}(L_m)) \subseteq \mathcal{L}(\text{MaxPlus}(L_q))$ and $\mathcal{L}(\text{Tree}(L_m)) \subseteq \mathcal{L}(\text{Tree}(L_q))$ (as stated in Lemma 4.10).

4.4 Composition and Constraining Operators

In Section 4.2 we defined equivalence and inclusion relations on logistics automata to reason about their behavior and state-space size. In this section we analyze the commutativity, distributivity and associativity properties of the composition and constraining operators with respect to the equivalence and inclusion relations.

The composition operator is both commutative and associative as claimed by the following lemma:

Lemma 4.11 — (Commutativity and Associativity). Let L_1, L_2 and L_3 be logistics automata. Then $L_1 \odot L_2 \approx L_2 \odot L_1$ and $(L_1 \odot L_2) \odot L_3 \approx L_1 \odot (L_2 \odot L_3)$.

Proof. We will only demonstrate commutativity; associativity is proven in a similar manner. Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ be logistics automata. Below we will show that for all $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$, $(s,t) \in \mathcal{S}_{L_1 \odot L_2}$ iff $(t,s) \in \mathcal{S}_{L_2 \odot L_1}$ and that for all $s_{0_1} \in \mathcal{S}_{0_1}$ and $s_{0_2} \in \mathcal{S}_{0_2}$, $(s_{0_1}, s_{0_2}) \in \mathcal{S}_{0_{L_1 \odot L_2}}$ iff $(s_{0_2}, s_{0_1}) \in \mathcal{S}_{0_{L_2 \odot L_1}}$ (where $\mathcal{S}_{L_1 \odot L_2}$ and $\mathcal{S}_{L_2 \odot L_1}$ denote the state-spaces of respectively $L_1 \odot L_2$ and $L_2 \odot L_1$ and where

$\mathcal{S}_{0_{L_1 \odot L_2}}$ and $\mathcal{S}_{0_{L_2 \odot L_1}}$ denote the sets of initial states of respectively $L_1 \odot L_2$ and $L_2 \odot L_1$). Now either i) $\mathcal{S}_{L_1 \odot L_2} = \emptyset$ or ii) $\mathcal{S}_{0_{L_1 \odot L_2}} = \{(s_{0_1}, s_{0_2})\}$ (where $s_{0_1} \in S_{0_1}$ and $s_{0_2} \in S_{0_2}$). In case i) $\mathcal{S}_{L_2 \odot L_1} = \emptyset$ and thus $L_1 \odot L_2 \approx L_2 \odot L_1$. For case ii) define $F((s,t)) = (t,s)$ for all $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$. F is a bijective function satisfying condition *1.* of Definition 4.1. Further $(s,t) \xrightarrow{a}_{L_1 \odot L_2} (s',t')$ iff $(t,s) \xrightarrow{a}_{L_2 \odot L_1} (t',s')$ (where $\dot{\rightarrow}_{L_1 \odot L_2}$ and $\dot{\rightarrow}_{L_2 \odot L_1}$ denote the transition relations of $L_1 \odot L_2$ and $L_2 \odot L_1$ respectively) and thus condition *2.* of Definition 4.1 is also satisfied. Hence $L_1 \odot L_2 \approx L_2 \odot L_1$.

The facts that $(s,t) \in \mathcal{S}_{L_1 \odot L_2}$ iff $(t,s) \in \mathcal{S}_{L_2 \odot L_1}$ and that $(s_{0_1}, s_{0_2}) \in \mathcal{S}_{0_{L_1 \odot L_2}}$ iff $(s_{0_2}, s_{0_1}) \in \mathcal{S}_{0_{L_2 \odot L_1}}$ follow straightforwardly from the following property: for all $(s,t),(s',t') \in \mathcal{S}_1 \times \mathcal{S}_2$ and $\underline{a} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^*$, $(s,t) \xrightarrow{\underline{a}}{}'_{L_1 \odot L_2} (s',t')$ iff $(t,s) \xrightarrow{\underline{a}}{}'_{L_2 \odot L_1} (t',s')$ (where $\dot{\rightarrow}'_{L_1 \odot L_2}$ and $\dot{\rightarrow}'_{L_2 \odot L_1}$ refer to the transitions relations of $L_1 \odot L_2$ and $L_2 \odot L_1$ before pruning). This property is proved by induction on the structure of $\underline{a}$:

- In case $\underline{a} = \epsilon$, $(s,t) \xrightarrow{\epsilon}{}'_{L_1 \odot L_2} (s',t')$ and thus $(s,t) = (s',t')$. But then also $(t,s) = (t',s')$ and $(t,s) \xrightarrow{\epsilon}{}'_{L_2 \odot L_1} (t',s')$.
- Otherwise $\underline{a} = \underline{b}a$ for some $\underline{b} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^*$ and $(s,t) \xrightarrow{\underline{b}a}{}'_{L_1 \odot L_2} (s',t')$. This implies $(s,t) \xrightarrow{\underline{b}}{}'_{L_1 \odot L_2} (s'',t'') \xrightarrow{a}{}'_{L_1 \odot L_2} (s',t')$ (for some $(s'',t'') \in \mathcal{S}_1 \times \mathcal{S}_2$). By induction we then have $(t,s) \xrightarrow{\underline{b}}{}'_{L_2 \odot L_1} (t'',s'')$. Showing that also $(t'',s'') \xrightarrow{a}{}'_{L_2 \odot L_1} (t',s')$ depends on whether inference rule (1), (2) or (3) of Definition 2.11 is used in the derivation:
  (1) In that case $a \in \mathcal{A}ct_1 \backslash \mathcal{A}ct_2$, $s'' \xrightarrow{a}_1 s'$ and $t'' = t'$. But then via inference rule (3) we have $(t'',s'') \xrightarrow{a}{}'_{L_2 \odot L_1} (t',s')$.
  (2) In that case $a \in \mathcal{A}ct_1 \cap \mathcal{A}ct_2$ then $s'' \xrightarrow{a}_1 s'$ and $t'' \xrightarrow{a}_2 t'$. But then via inference rule (2) we have $(t'',s'') \xrightarrow{a}{}'_{L_2 \odot L_1} (t',s')$.
  (3) In that case $a \in \mathcal{A}ct_2 \backslash \mathcal{A}ct_1$, $t'' \xrightarrow{a}_2 t'$ and $s'' = s'$. But then via inference rule (1) we have $(t'',s'') \xrightarrow{a}{}'_{L_2 \odot L_1} (t',s')$.

■

Another very useful property is that our equivalence and inclusion relations are substitutive under the composition operator:

Lemma 4.12 — (Substitutivity 3). Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ and $L = \langle \mathcal{S}, \mathcal{A}ct, \dot{\rightarrow}, S_0 \rangle$ be logistics automata. Then $L_1 \sqsubseteq L_2$ implies $L_1 \odot L \sqsubseteq L_2 \odot L$ and $L_1 \approx L_2$ implies $L_1 \odot L \approx L_2 \odot L$.

Proof. Substitutivity of $\approx$ follows directly from the substitutivity of $\sqsubseteq$. To prove substitutivity of $\sqsubseteq$ assume $L_1 \sqsubseteq L_2$. Firstly remark that $\mathcal{A}ct_{L_1 \odot L} = \mathcal{A}ct_{L_2 \odot L}$. Further let $\mathcal{S}_{L_1 \odot L}$ and $\mathcal{S}_{L_2 \odot L}$ denote the state-spaces of $L_1 \odot L$ and $L_2 \odot L$ respectively and distinguish two cases. Either i) $\mathcal{S}_{L_1 \odot L} = \emptyset$ or ii) $\mathcal{S}_{0_{L_1 \odot L}} = \{(s_{0_1}, s_0)\}$ (where $s_{0_1} \in S_{0_1}$ and $s_0 \in S_0$). In case i) obviously $L_1 \odot L \sqsubseteq L_2 \odot L$. In case ii) an injective relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ exists satisfying conditions *1., 2. and 3.* of Definition 4.2. Based on R we define a new relation $R' = \{((s_1, s), (s_2, s)) \mid (s_1, s) \in \mathcal{S}_{L_1 \odot L}, (s_2, s) \in \mathcal{S}_{L_2 \odot L} \text{ and } (s_1, s_2) \in R\}$. Since R is injective, R' is injective as well. We still have to show that R' satisfies conditions *1., 2. and 3.* of Definition 4.2. For this let $\dot{\rightarrow}'_{L_2 \odot L}$ denote the transition relation of $L_2 \odot L$ *before pruning* (corresponding to relation $\dot{\rightarrow}$ in Definition 2.11).

For condition *1.* we know $(s_{0_1}, s_0) \in \mathcal{S}_{L_1 \odot L}$ which implies that $(s_{0_1}, s_0) \dot{\rightarrow}^*_{L_1 \odot L} (s_1, s)$ for some $s_1 \in \mathcal{S}_1$ and $s \in \mathcal{S}$ for which $s_1 \not\rightarrow_1$ and $s \not\dot{\rightarrow}$. Since $(s_{0_1}, s_{0_2}) \in R$, $(s_{0_2}, s_0) \dot{\rightarrow}'^*_{L_2 \odot L} (s_2, s)$ for some $s_2 \in \mathcal{S}_2$ such that $(s_1, s_2) \in R$ (which can be proven by induction on the number of steps to transit from (s_{0_1}, s_0) to (s_1, s) distinguishing the different inference rules in Definition 2.11). Now $s_2 \not\rightarrow_2$ since $s_1 \not\rightarrow_1$ and hence $(s_{0_2}, s_0) \in \mathcal{S}_{L_2 \odot L}$. Therefore $((s_{0_1}, s_0), (s_{0_2}, s_0)) \in R'$ and thus condition *1.* is satisfied.

For condition *2.* assume $((s_1, s), (s_2, s)) \in R'$, $a \in \mathcal{A}ct_{L_1 \odot L}$ and $(s_1, s) \xrightarrow{a}_{L_1 \odot L} (s'_1, s')$ for some $(s'_1, s') \in \mathcal{S}_{L_1 \odot L}$. We have to show that $(s_2, s) \xrightarrow{a}_{L_2 \odot L} (s'_2, s')$ for some $(s'_2, s') \in \mathcal{S}_{L_2 \odot L}$ such that $((s'_1, s'), (s'_2, s')) \in R'$. For this we distinguish three cases, dependent on which inference rule of Definition 2.11 was used to derive $(s_1, s) \xrightarrow{a}_{L_1 \odot L} (s'_1, s')$:

(1) In case rule (1) was used $a \in \mathcal{A}ct_1 \backslash \mathcal{A}ct$ and $s_1 \xrightarrow{a}_1 s'_1$ and $s' = s$. Since $(s_1, s_2) \in R$ we know $s_2 \xrightarrow{a}_2 s'_2$ for some $s'_2 \in \mathcal{S}_2$ such that $(s'_1, s'_2) \in R$. Since $a \in \mathcal{A}ct_2 \backslash \mathcal{A}ct$ rule (1) can be applied again to derive $(s_2, s) \xrightarrow{a}'_{L_2 \odot L} (s'_2, s)$. To see that $(s'_2, s) \in \mathcal{S}_{L_2 \odot L}$ first notice that $(s_{0_2}, s_0) \dot{\rightarrow}'^*_{L_2 \odot L} (s'_2, s)$. Further since $(s'_1, s) \in \mathcal{S}_{L_1 \odot L}$ we know that $(s'_1, s) \dot{\rightarrow}^*_{L_1 \odot L} (s''_1, s'')$ for some $s''_1 \in \mathcal{S}_1$ and $s'' \in \mathcal{S}$ for which $s''_1 \not\rightarrow_1$ and $s'' \not\dot{\rightarrow}$. Now since $(s'_1, s'_2) \in R$, $(s'_2, s) \dot{\rightarrow}'^*_{L_2 \odot L} (s''_2, s'')$ for some

$s_2'' \in \mathcal{S}_2$ such that $s_2'' \not\dot{\rightarrow}_2$ (which can be proven by induction on the number of steps to transit from (s_1', s) to (s_1'', s'') distinguishing the different inference rules in Definition 2.11). Therefore $(s_2', s) \in \mathcal{S}_{L_2 \odot L}$ which implies $(s_2, s) \xrightarrow{a}_{L_2 \odot L} (s_2', s)$. Hence also $((s_1', s), (s_2', s)) \in R'$ holds.

(2) In case rule (2) was used $a \in \mathcal{A}ct_1 \cap \mathcal{A}ct$ and $s_1 \xrightarrow{a}_1 s_1'$ and $s \xrightarrow{a} s'$. Since $(s_1, s_2) \in R$ we know $s_2 \xrightarrow{a}_2 s_2'$ for some $s_2' \in \mathcal{S}_2$ such that $(s_1', s_2') \in R$. Since $a \in \mathcal{A}ct_2 \cap \mathcal{A}ct$ rule (2) can be applied again to derive $(s_2, s) \xrightarrow{a}'_{L_2 \odot L} (s_2', s')$. The proof that $(s_2', s') \in \mathcal{S}_{L_2 \odot L}$ and $((s_1', s'), (s_2', s')) \in R'$ is the same as in case (1).

(3) In case rule (3) was used $a \in \mathcal{A}ct \backslash \mathcal{A}ct_1$ and $s_1 = s_1'$ and $s \xrightarrow{a} s'$. Since $a \in \mathcal{A}ct \backslash \mathcal{A}ct_2$ rule (3) can be applied again to derive $(s_2, s) \xrightarrow{a}'_{L_2 \odot L} (s_2, s')$. The proof that $(s_2, s') \in \mathcal{S}_{L_2 \odot L}$ and $((s_1, s'), (s_2, s')) \in R'$ is the same as in case (1).

For condition *3.* assume $((s_1, s), (s_2, s)) \in R'$, and $(s_2, s) \dot{\rightarrow}_{L_2 \odot L}$. This implies that either $s_2 \dot{\rightarrow}_2$ or $s \dot{\rightarrow}$. Since $(s_1, s_2) \in R$ we then know that either $s_1 \dot{\rightarrow}_1$ or $s \dot{\rightarrow}$. Since $(s_1, s) \in \mathcal{S}_{L_1 \odot L}$, $(s_1, s) \dot{\rightarrow}^*_{L_1 \odot L} (s_1', s')$ for some $s_1' \in \mathcal{S}_1$ and $s' \in \mathcal{S}$ for which $s_1' \not\dot{\rightarrow}_1$ and $s' \not\dot{\rightarrow}$. So $(s_1, s) \not\dot{\rightarrow}_{L_1 \odot L}$ implies $s_1 \not\dot{\rightarrow}_1$ and $s \not\dot{\rightarrow}$. Vice versa $s_1 \dot{\rightarrow}_1$ or $s \dot{\rightarrow}$ implies $(s_1, s) \dot{\rightarrow}_{L_1 \odot L}$. Hence $(s_1, s) \dot{\rightarrow}_{L_1 \odot L}$.

■

Example 4.7 Consider the logistics automata L_1, L_2 and L in Figure 4.5. L_1 is included in L_2 and because of this $L_1 \odot L$ is included in $L_2 \odot L$ (as is claimed in Lemma 4.12).

Example 4.8 Consider the logistics automata L_1, L_2 and L in Figure 4.6, where L_1 is included in L_2. The resulting composite automata $L_1 \odot L$ and $L_2 \odot L$ are both empty and thus $L_1 \odot L$ is included in $L_2 \odot L$ indeed. Here we notice that this substitutivity result (Lemma 4.12) would not hold if we had required $\mathcal{A}ct_1 \subseteq \mathcal{A}ct_2$ instead of $\mathcal{A}ct_1 = \mathcal{A}ct_2$ in Definition 4.2; the proof of the lemma would fail for condition 2), inference rule 3. For instance for $\mathcal{A}ct_1 = \{a, b\}$ and $\mathcal{A}ct_2 = \{a, b, c\}$, L_1 would then be included in L_2 but $L_1 \odot L$ has language $\{abc\}$ while $L_2 \odot L$ is empty.

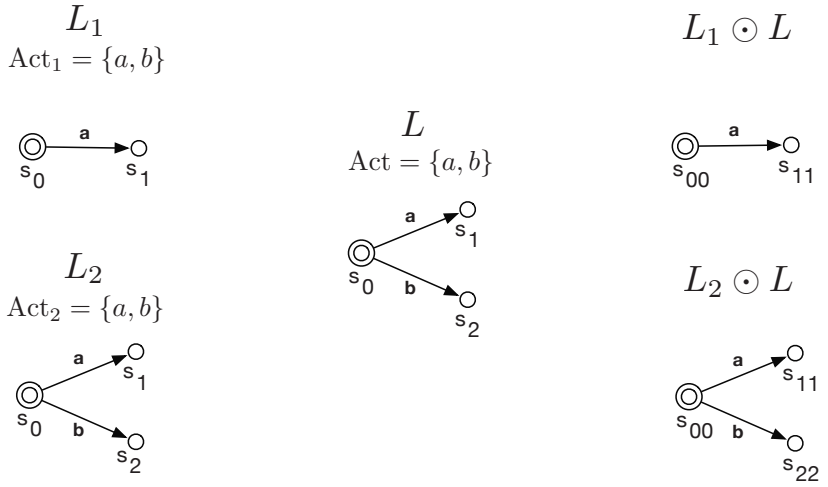


Figure 4.5: *Substitutivity example where composed automaton $L_1 \odot L$ is included in composed automaton $L_2 \odot L$.*

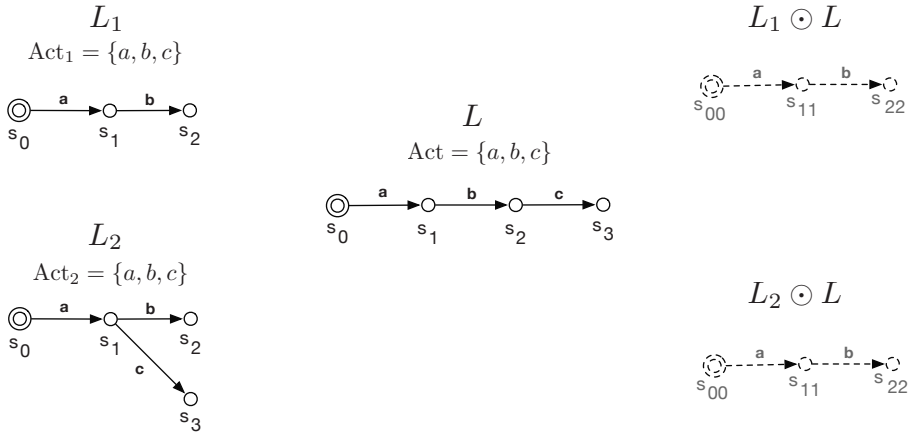


Figure 4.6: *Substitutivity example where composed automaton $L_1 \odot L$ is only included in composed automaton $L_2 \odot L$ because $\mathcal{A}ct_1 = \mathcal{A}ct_2$.*

From Lemma 2.1 it follows straightforwardly that both language inclusion and language equivalence are substitutive under the composition operator:

Lemma 4.13 — (Substitutivity 4). Let L_1, L_2 and L be logistics automata. Then $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ implies $\mathcal{L}(L_1 \odot L) \subseteq \mathcal{L}(L_2 \odot L)$ and $\mathcal{L}(L_1) = \mathcal{L}(L_2)$ implies $\mathcal{L}(L_1 \odot L) = \mathcal{L}(L_2 \odot L)$.

Example 4.9 Consider the logistics automata L_1, L_2 and L in Figure 4.5 again. Since $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ we also have $\mathcal{L}(L_1 \odot L) \subseteq \mathcal{L}(L_2 \odot L)$ (as is claimed in Lemma 4.13). Also consider the logistics automata L_1, L_2 and L in Figure 4.6. Since $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ we also have $\mathcal{L}(L_1 \odot L) \subseteq \mathcal{L}(L_2 \odot L)$ (as is claimed in Lemma 4.13). Here we notice that this substitutivity results would not hold if we had required $\mathcal{A}ct_1 \subseteq \mathcal{A}ct_2$ instead of $\mathcal{A}ct_1 = \mathcal{A}ct_2$ in Definition 4.2; even though Lemma 2.1 would still hold, Lemma 4.13 would not. For instance for $\mathcal{A}ct_1 = \{a, b\}$ and $\mathcal{A}ct_2 = \{a, b, c\}$, L_1 would be included in L_2 but $L_1 \odot L$ has language $\{abc\}$ while $L_2 \odot L$ is empty.

Constraining is a commutative operator and distributes over automaton composition, which is claimed in the following lemmas:

Lemma 4.14 — (Commutativity). Let L be a logistics automaton and let C_1 and C_2 be two constraints on L. Then

$$(L \upharpoonright C_1) \upharpoonright C_2 \approx (L \upharpoonright C_2) \upharpoonright C_1$$

Proof. The proof is similar to the proof of Lemma 4.11. ■

Lemma 4.15 — (Distributivity). Let L_1 and L_2 be two logistics automata and C be a constraint on both L_1 and on L_2. Then

$$(L_1 \odot L_2) \upharpoonright C \approx (L_1 \upharpoonright C) \odot (L_2 \upharpoonright C)$$

Proof. The proof is similar to the proof of Lemma 4.11.

■

Another very useful property is that our equivalence and inclusion relations are substitutive under the constraint operator:

Lemma 4.16 — (Substitutivity 5). Let L_1 and L_2 be logistics automata and let C be a constraint on L_1 and L_2. Then $L_1 \sqsubseteq L_2$ implies $L_1 \restriction C \sqsubseteq L_2 \restriction C$ and $L_1 \approx L_2$ implies $L_1 \restriction C \approx L_2 \restriction C$.

Proof. The proof is similar to the proof of Lemma 4.12. ■

From Lemma 2.2 it follows straightforwardly that both language inclusion and language equivalence are substitutive under the constraint operator:

Lemma 4.17 — (Substitutivity 6). Let L_1, L_2 be logistics automata and let C be a constraint on both L_1 and L_2. Then $\mathcal{L}(L_1) \subseteq \mathcal{L}(L_2)$ implies $\mathcal{L}(L_1 \restriction C) \subseteq \mathcal{L}(L_2 \restriction C)$ and $\mathcal{L}(L_1) = \mathcal{L}(L_2)$ implies $\mathcal{L}(L_1 \restriction C) = \mathcal{L}(L_2 \restriction C)$.

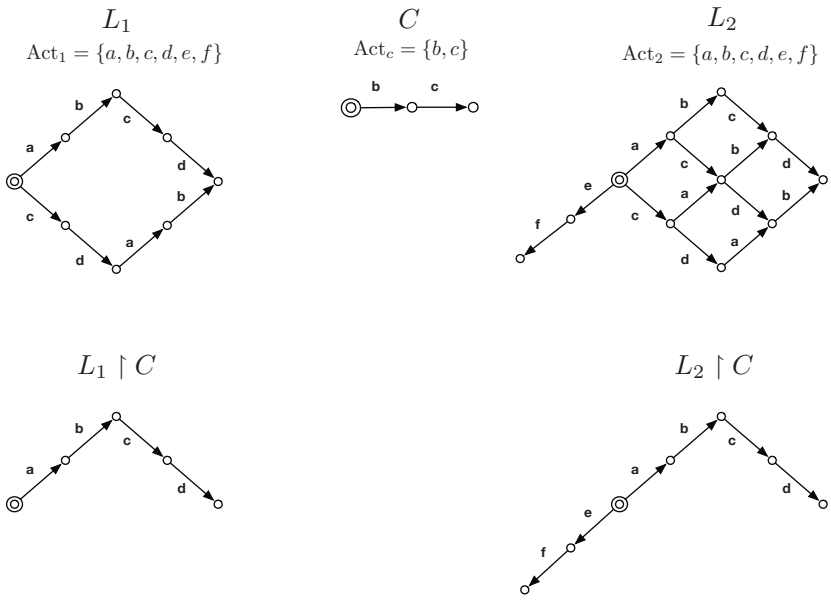


Figure 4.7: *Example of the substitutivity property in the constraining operation using logistics automata L_1 and L_2, constraint automaton C and the resulting constrained automata $L_1 \restriction C$ and $L_2 \restriction C$.*

Example 4.10 Consider logistics automata L_1 and L_2 and constraint automata C depicted in Figure 4.7. First notice that $L_1 \sqsubseteq L_2$ and $\mathcal{A}ct_{L_1} = \mathcal{A}ct_{L_2}$ and $\mathcal{A}ct_C \subseteq \mathcal{A}ct_{L_1}$. Consider constrained automata $L_1 \restriction C$ and $L_2 \restriction C$ also depicted in Figure 4.7. Note that $L_1 \restriction C \sqsubseteq L_2 \restriction C$ which should indeed be the case according to Lemma 4.16.

4.5 Optimization-space Reduction

In this section we discuss how to reduce[1] the optimization-space by constraining logistics automata. For this purpose we establish sufficient conditions on L and C so that $L \restriction C \sqsubseteq L$. First note that the application of a constraint to a logistics automaton, results in a subset of the original language. This follows immediately from Lemma 2.2:

Lemma 4.18 — (Language constraining). Let L be a logistics automaton and let C be a constraint on L. Then $\mathcal{L}(L \restriction C) \subseteq \mathcal{L}(L)$.

Example 4.11 Consider again logistics automata L_1 and L_2 and constraint automata C depicted in Figure 4.7. Notice that $\mathcal{L}(L_1) = \{abcd, cdab\}$, $\mathcal{L}(L_1 \restriction C) = \{abcd\}$ and $\mathcal{L}(L_2 \restriction C) = \{abcd, ef\}$ and that $\mathcal{L}(L_1 \restriction C) \subseteq \mathcal{L}(L_2 \restriction C)$ (which should indeed be the case according to Lemma 4.5 since $L_1 \restriction C \sqsubseteq L_2 \restriction C$). Further observe that $\mathcal{L}(L_1 \restriction C) \subseteq \mathcal{L}(L_1)$ (which follows from Lemma 4.18).

Lemma 2.3 tells us that the application of a constraint will reduce the language induced by the constrained automaton. This does not imply however that $L \restriction C$ is included in L. In general even MaxPlus$(L \restriction C)$ is not included in MaxPlus(L) and Tree$(L \restriction C)$ is not included in 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 (see Example 4.12).

Example 4.12 Consider logistics automaton L_k , constraint automata C_1 and $L_k \restriction C_1$ depicted in Figure 4.8. Obviously $L_k \restriction C_1 \not\sqsubseteq L_k$, MaxPlus$(L_k \restriction C) \not\subseteq$ MaxPlus(L_k) and Tree$(L_k \restriction C) \not\subseteq$ Tree(L_k). This is caused by constraint C_1 which is non-deterministic.

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

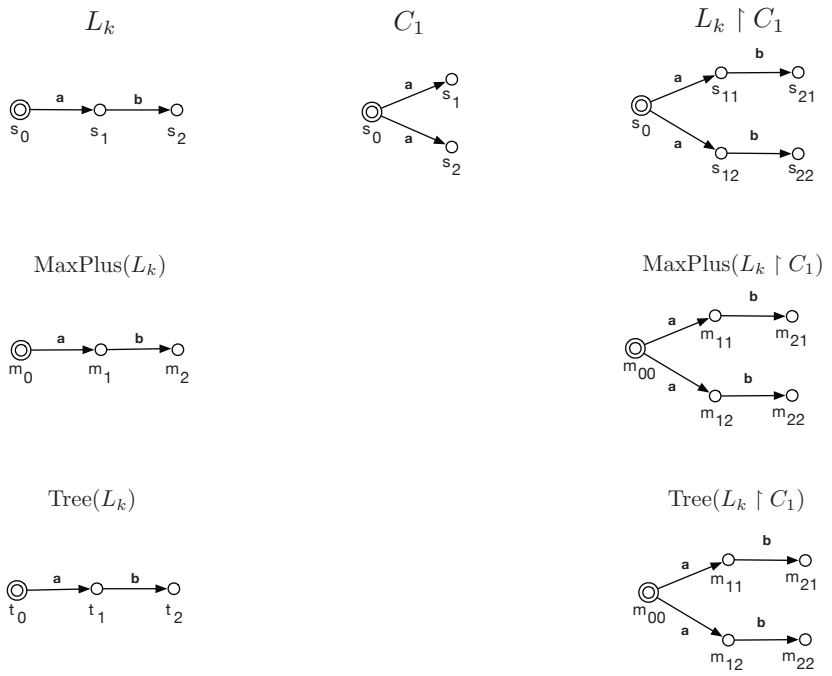


Figure 4.8: *Example showing that the constraining of L_k with a non-deterministic constraint C_1 results in a constrained automaton $L_k \restriction C$ which is not included in L_k. The same holds true for their $(max, +)$ and tree counterparts.*

Example 4.12 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 4.19 Let $L = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ be a logistics automaton and $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ a constraint on L. If C is deterministic, then $\text{Tree}(L \restriction C) \sqsubseteq \text{Tree}(L)$.

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

$$\frac{S_{0_{\text{Tree}(L\restriction C)}} = \{(s_0, c_0)\}}{((s_0, c_0), s_0) \in V} \ (1) \qquad \frac{\big(q\ (s,c), p\ s\big) \in V \quad (s,c) \xrightarrow{a}_{L\restriction C} (s', c')}{\big(q\ (s,c)\ a\ (s', c'), p\ s\ a\ s'\big) \in V} \ (2)$$

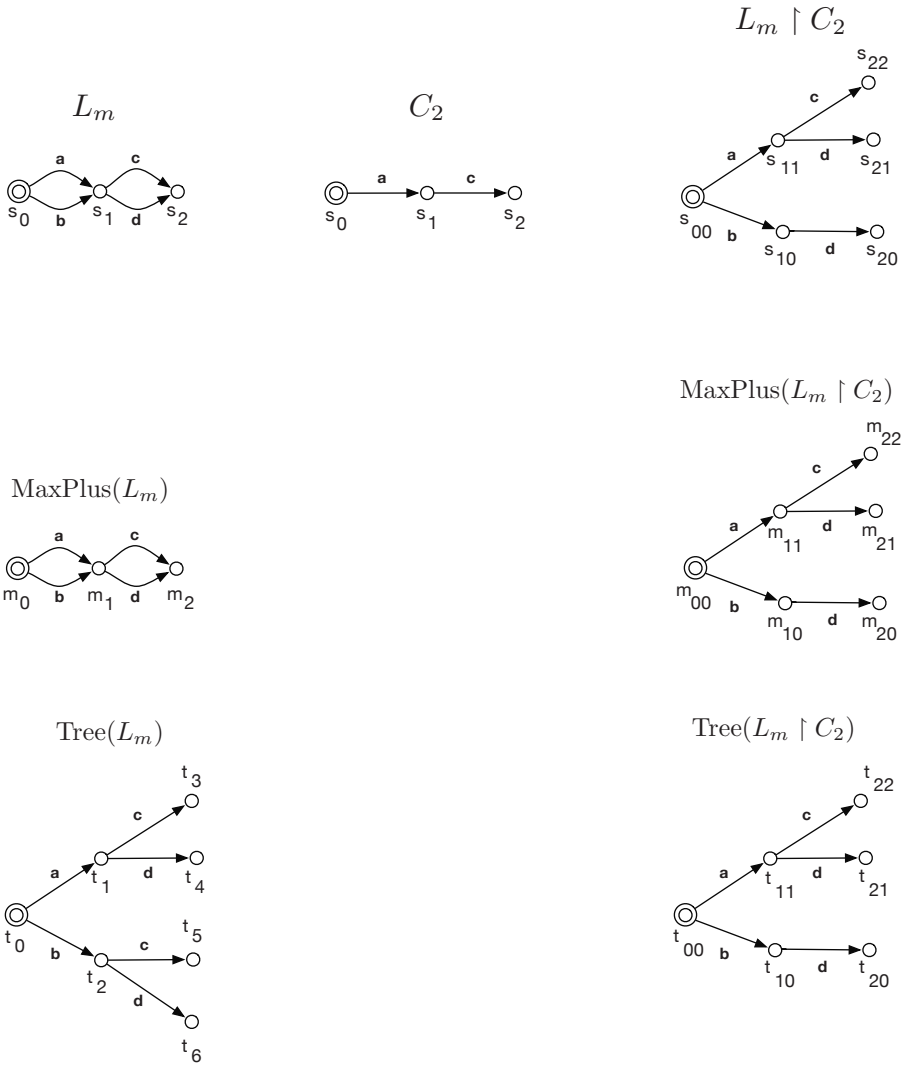
Recall that $q\ (s,c)$ and $p\ s$ refer to a paths that end in states $(s,c) \in \mathcal{S}_{L\restriction C}$ and $s \in \mathcal{S}_1$ respectively. We have to show that R is injective and satisfies condition *1.*, *2.* and *3.* of Definition 4.2. Now clearly $((s_0, c_0), s_0) \in R$ so condition 1. is satisfied. For condition 2. let $\big(q\ (s,c), p\ s\big) \in R$ and assume $q\ (s,c) \xrightarrow{a}_{\text{Tree}(L\restriction C)} q'$ (for some $a \in \mathcal{A}ct_1$ and $q' \in \text{Paths}(L \restriction C)$). Then q' must be of the form $q\ (s,c)\ a\ (s', c')$ for some $(s', c') \in \mathcal{S}_{L\restriction C}$ for which $(s,c) \xrightarrow{a}_{L\restriction C} (s', c')$. But then $s \xrightarrow{a}_1 s'$ and thus $p\ s \xrightarrow{a}_{\text{Tree}(L)} p\ s\ a\ s'$. Consequently $\big(q\ (s,c)\ a\ (s', c'), p\ s\ a\ s'\big) \in R$, which follows from inference rule (2). For condition 3. let $\big(q\ (s,c), p\ s\big) \in R$ and assume $p\ s \dot{\rightarrow}_{\text{Tree}(L)}$. Then $s \dot{\rightarrow}_1$ and since $(s,c) \in \mathcal{S}_{L\restriction C}$ also $(s,c) \dot{\rightarrow}_{L\restriction C}$. But then also $q\ (s,c) \dot{\rightarrow}_{\text{Tree}(L\restriction C)}$.

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

Example 4.13 Consider logistics automaton L_m, constraint C_2, and constrained automaton $L_m \restriction C_2$ depicted in Figure 4.9. Since C_2 is deterministic we have $\text{Tree}(L_m \restriction C) \sqsubseteq \text{Tree}(L_m)$ (consistent with Lemma 4.19). On the other hand, we also see that $L_m \restriction C \not\sqsubseteq L_m$ and that $\text{MaxPlus}(L_m \restriction C) \not\sqsubseteq \text{MaxPlus}(L_m)$. Notice in particular that constraining has led to an increased size of the (max,+) state-space.

4.5.1 Nonpermutation-repulsing, Permutation-attracting and Confluent Automata

From Lemma 4.19 and Example 4.13 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



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

Figure 4.9: *Example showing that in the case of a deterministic constraint C_2 the tree of the constrained automaton (Tree($L_m \restriction C_2$)) 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.*

Lemma 4.9 stating that $L \restriction C \sqsubseteq L$ implies MaxPlus$(L \restriction C) \sqsubseteq$ MaxPlus(L). We thus want to make sure that $L \restriction C \sqsubseteq L$. In the remainder of this section we will establish sufficient conditions (on L and C) for this to become true. To this end we start by defining the two fundamental properties of automata.

Definition 4.4 — (Nonpermutation-repulsing). 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 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 $\underline{a} \sim^p \underline{a}'$.

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 4.5 — (Permutation-attracting). 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 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 4.14 Consider the automata L_q, L_r, L_s and L_t of Figure 4.10. 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 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.

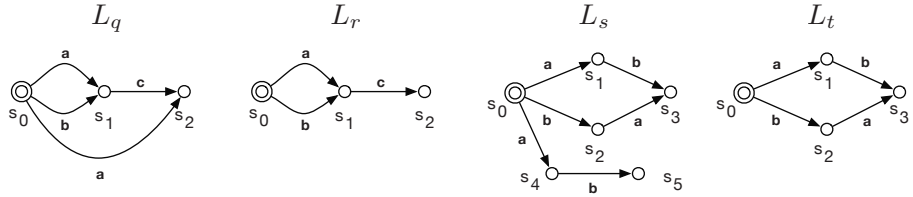


Figure 4.10: *Examples of automata: L_q is neither np-repulsing nor p-attracting. L_r is p-attracting but not np-repulsing. L_s is np-repulsing but not p-attracting. L_t is both np-repulsing and p-attracting.*

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

Lemma 4.20 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 $L_1 \odot L_2$.

Proof. Let $L_1 = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ and $L_2 = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$ be logistics automata. If either $\mathcal{S}_1 = \emptyset$ or $\mathcal{S}_2 = \emptyset$ then $\mathcal{S}_{L_1 \odot L_2} = \emptyset$ implying that $L_1 \odot L_2$ is both np-repulsing and p-attracting. Otherwise $S_{0_1} = \{s_{0_1}\}$ and $S_{0_2} = \{s_{0_2}\}$ (for some $s_{0_1} \in \mathcal{S}_1$ and $s_{0_2} \in \mathcal{S}_2$). We have to show that condition *1.* of Definition 4.4 and Definition 4.5 are preserved by the composition operator.

For condition *1.* of Definition 4.4 assume $(s_1, s_2) \in \mathcal{S}_{L_1 \odot L_2}$ and let $\underline{a}, \underline{a'} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^*$. Assume $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{a}}_{L_1 \odot L_2} (s_1, s_2)$ and $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{a'}}_{L_1 \odot L_2} (s_1, s_2)$. Then (see also the proof of Lemma 2.1) $s_{0_1} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_1}_1 s_1$, $s_{0_1} \xrightarrow{\underline{a'} \backslash \mathcal{A}ct_1}_1 s_1$, $s_{0_2} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_2}_2 s_1$ and $s_{0_2} \xrightarrow{\underline{a'} \backslash \mathcal{A}ct_2}_2 s_2$. But then (if L_1 and L_2 satisfy condition *1.* of Definition 4.4) $\underline{a} \backslash \mathcal{A}ct_1 \sim^p \underline{a'} \backslash \mathcal{A}ct_1$ and $\underline{a} \backslash \mathcal{A}ct_2 \sim^p \underline{a'} \backslash \mathcal{A}ct_2$. This implies that $\underline{a}$ and $\underline{a'}$ contain the same number of activities in $\mathcal{A}ct_1 \backslash \mathcal{A}ct_2$, the same number of activities in $\mathcal{A}ct_1 \cap \mathcal{A}ct_2$ and the same number of activities in $\mathcal{A}ct_2 \backslash \mathcal{A}ct_1$. But then $\underline{a} \sim^p \underline{a'}$ and thus condition *1.* of Definition 4.4 holds.

For condition *1.* of Definition 4.5 let $(s_1, s_2), (s'_1, s'_2) \in \mathcal{S}_{L_1 \odot L_2}$ and let $\underline{a}, \underline{a'} \in (\mathcal{A}ct_1 \cup \mathcal{A}ct_2)^*$ such that $\underline{a} \sim^a \underline{a'}$. Assume $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{a}}_{L_1 \odot L_2} (s_1, s_2)$ and $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{a'}}_{L_1 \odot L_2} (s'_1, s'_2)$. Then $s_{0_1} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_1}_1 s_1$, $s_{0_1} \xrightarrow{\underline{a} \backslash \mathcal{A}ct_1}_1 s'_1$,

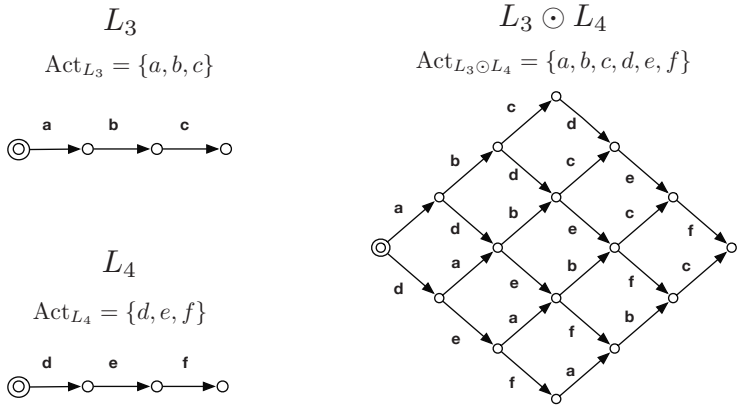


Figure 4.11: *The composition of two np-repulsing and p-attracting automata is again a np-resulsing and p-attracting automaton.*

$s_{0_2} \xrightarrow{\underline{a}\backslash \mathcal{A}ct_2}_2 s_2$ and $s_{0_2} \xrightarrow{\underline{a}\backslash \mathcal{A}ct_2}_2 s_2'$. Since $\underline{a}\backslash \mathcal{A}ct_1 \sim^p \underline{a}'\backslash \mathcal{A}ct_1$ and $\underline{a}\backslash \mathcal{A}ct_2 \sim^p \underline{a}'\backslash \mathcal{A}ct_2$ we have $s_1 = s_1'$ and $s_2 = s_2'$ (if L_1 and L_2 satisfy condition *1.* of Definition 4.5). Hence $(s_1, s_2) = (s_1', s_2')$. ■

Example 4.15 Consider logistics automata L_3, L_4 and $L_3 \odot L_4$ depicted in Figure 4.11. Both L_3 and L_4 are np-repulsing and p-attracting and so is $L_3 \odot L_4$ (consistent with Lemma 4.20).

Lemma 4.21 Let L be an np-repulsing logistics automaton and let C be any constraint. Then $L \restriction C$ is np-repulsing.

Proof. 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. If either $\mathcal{S}_1 = \emptyset$ or $\mathcal{S}_2 = \emptyset$ then $\mathcal{S}_{L\restriction C} = \emptyset$ implying that $L \restriction C$ is np-repulsing. Otherwise $S_{0_1} = \{s_{0_1}\}$ and $S_{0_2} = \{s_{0_2}\}$ (for some $s_{0_1} \in \mathcal{S}_1$ and $s_{0_2} \in \mathcal{S}_2$). We have to show that condition *1.* of Definition 4.4 is preserved by the constraint operator. To this end assume $(s_1, s_2) \in \mathcal{S}_{L\restriction C}$ and let $\underline{a}, \underline{a'} \in \mathcal{A}ct_1^*$ (notice that $\mathcal{A}ct_2 \subseteq \mathcal{A}ct_1$). Assume $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{a}}_{L\restriction C} (s_1, s_2)$ and $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{a}'}_{L\restriction C} (s_1, s_2)$. Then (see also the proof of Lemma 2.2) $s_{0_1} \xrightarrow{\underline{a}}_1 s_1$, $s_{0_1} \xrightarrow{\underline{a}'}_1 s_1$, $s_{0_2} \xrightarrow{\underline{a}\backslash \mathcal{A}ct_2}_2 s_2$ and $s_{0_2} \xrightarrow{\underline{a}'\backslash \mathcal{A}ct_2}_2 s_2$. Therefore (if L is np-repulsing) $\underline{a} \sim^p \underline{a}'$ and thus condition *1* is preserved (irrespective of whether C is np-repulsing or not). ■

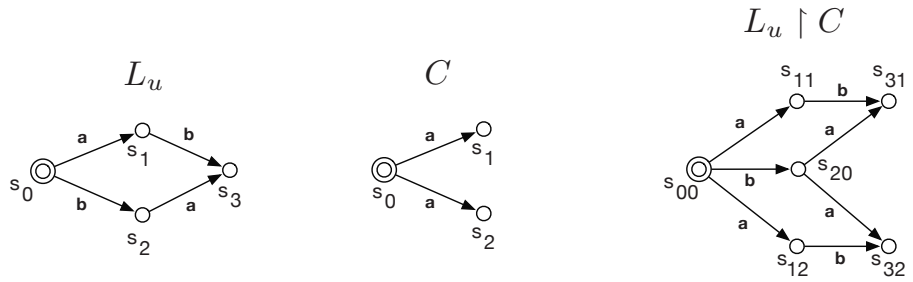


Figure 4.12: *Np-repulsiveness is preserved by constraining; p-attractiveness is not.*

Example 4.16 Consider automaton L_u and constraint C depicted in Figure 4.12. 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 4.21), but it is not p-attracting.

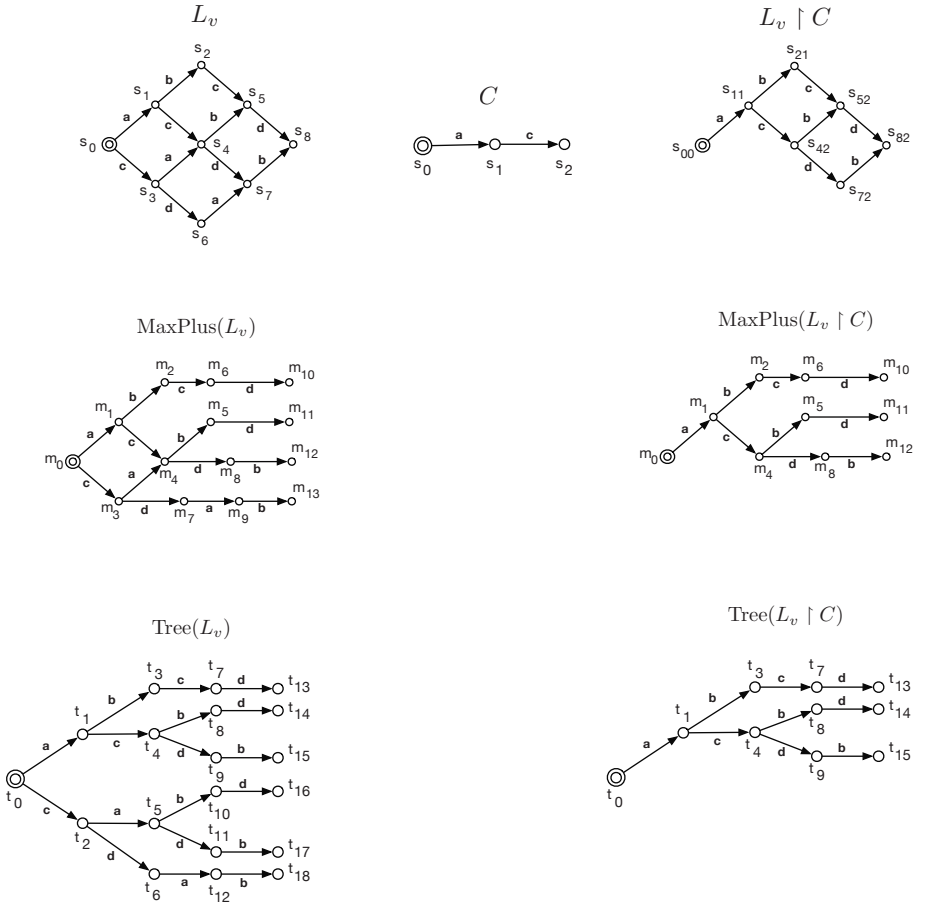
From the previous example we learn that p-attractiveness is not preserved by general constraining. However, if both the logistics automaton and the constraint are p-attractive, the constrained automaton is p-attractive as well, which is stated in the following lemma.

Lemma 4.22 Let L be a p-attracting logistics automaton and let C be a p-attracting constraint. Then $L \upharpoonright C$ is p-attracting.

Proof. The proof is similar to the proofs of Lemmas 4.20 and 4.21. ■

Example 4.17 Consider automaton L_u and constraint C depicted in Figure 4.12. 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. Further consider logistics automata L_1 and L_2 and constraint C depicted in Figure 4.7. L_1, L_2 and C are all p-attracting and so are $L_1 \upharpoonright C$ and $L_2 \upharpoonright C$.

We have now arrived at the main theorem of this chapter. 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.



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

Figure 4.13: *Constraining an np-repulsing automaton with a p-attracting constraint reduces the (max,+) optimization-space as well as the worst-case optimization-space.*

Theorem 4.23 Let L be an np-repulsing logistics automata and let C be a p-attracting constraint. Then $L \restriction C \sqsubseteq L$.

Proof. Let $L = \langle \mathcal{S}_1, \mathcal{A}ct_1, \dot{\rightarrow}_1, S_{0_1} \rangle$ and $C = \langle \mathcal{S}_2, \mathcal{A}ct_2, \dot{\rightarrow}_2, S_{0_2} \rangle$. If $\mathcal{S}_{L\restriction C} = \emptyset$ then $L \restriction C \sqsubseteq L$ follows immediately from Definition 4.2 (noticing that $\mathcal{A}ct_{L\restriction C} = \mathcal{A}ct_1$). Otherwise $S_{0_1} = \{s_{0_1}\}$, $S_{0_2} = \{s_{0_2}\}$ (for some $s_{0_1} \in \mathcal{S}_1$ and $s_{0_2} \in \mathcal{S}_2$) and $S_{0_{L\restriction C}} = \{(s_{0_1}, s_{0_2})\}$. Define relation $R \subseteq \mathcal{S}_{L\restriction C} \times \mathcal{S}_1$ as

$$R = \{((s_1, s_2), s_1) \mid (s_1, s_2) \in \mathcal{S}_{L\restriction C}\}$$

Notice that $(s_1, s_2) \in \mathcal{S}_{L\restriction C}$ implies $s_2 \in \mathcal{S}_1$. Using Definition 2.14 it is straightforward to show that R satisfies conditions *1.*, *2.* and *3.* of Definition 4.2. Remains us to show that R is injective. To this end let $((s_1, s_2), s_1), ((s_1, s_2'), s_1) \in R$. Then for some $\underline{a}, \underline{a}' \in \mathcal{A}ct_1^*$, $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{a}}_{L\restriction C} (s_1, s_2)$ and $(s_{0_1}, s_{0_2}) \xrightarrow{\underline{a}'}_{L\restriction C} (s_1, s_2')$. Therefore $s_{0_1} \xrightarrow{\underline{a}}_1 s_1, s_{0_1} \xrightarrow{\underline{a}'}_1 s_1, s_{0_2} \xrightarrow{\underline{a}\backslash\mathcal{A}ct_2}_2 s_2$ and $s_{0_2} \xrightarrow{\underline{a}'\backslash\mathcal{A}ct_2}_2 s_2'$. Since L is np-repulsing we thus know that $\underline{a} \sim^p \underline{a}'$. As a consequence also $\underline{a}\backslash\mathcal{A}ct_2 \sim^p \underline{a}'\backslash\mathcal{A}ct_2$. But then, since C is p-attracting, $s_2 = s_2'$. Thus R is injective. ■

Example 4.18 Consider the np-repulsing logistics automaton L_v and the p-attracting constraint C as shown in Figure 4.13. Consistent with Theorem 4.23 we observe that $L_v \restriction C \sqsubseteq L_v$. Further from Lemma 4.9 we know that both MaxPlus$(L_v \restriction C) \sqsubseteq$ MaxPlus(L_v) and Tree$(L_v \restriction C) \sqsubseteq$ Tree(L_v). Consequently from Lemma 4.4 we know that the both the (max,+) optimization space and the worst-case optimization space are reduced, which is also obvious from the picture.

The reader may have wondered about the relations between np-repulsiveness and p-attractiveness and the well-known concept of confluence. Indeed the automaton $L_3 \odot L_4$ depicted in Figure 4.11 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 4.23 applies when both the logistics automaton and the constraint automaton are confluent.

Before we give the necessary proofs and counter-examples, we will formalize the notion of confluence. Our definition is based on Milner's work on CCS [59] in which confluence is defined for labeled transition systems. To this end, we first define (following the definition in [59]) 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 4.6 — (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 properties that we require later in our proofs.

Lemma 4.24 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 sequence $\underline{b}$
(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) $(\underline{b}/\underline{c})/(\underline{c}/\underline{b}) = \underline{b}/\underline{c}$

Proof. The proofs of (1) and (3) are by induction on the structure of b. For (2) we have $\underline{b} \sim^p \underline{c}$ iff $\#_a(\underline{b}) = \#_a(\underline{c})$ (for all $a \in \mathcal{A}ct$) iff $max(\#_a(\underline{b}) - \#_a(\underline{c}), 0) = 0$ and $max(\#_a(\underline{c}) - \#_a(\underline{b}), 0) = 0$ (for all $a \in \mathcal{A}ct$) iff $\#_a(\underline{b}/\underline{c}) = 0$ and $\#_a(\underline{c}/\underline{b}) = 0$ (for all $a \in \mathcal{A}ct$) iff $\underline{b}/\underline{c} = \epsilon$ and $\underline{c}/\underline{b} = \epsilon$. For (4) we notice that if $\underline{b}/\underline{c}$ contains activity a (i.e. $\#_a(\underline{b}/\underline{c}) > 0$) then $\underline{c}/\underline{b}$ does not (i.e. $\#_a(\underline{c}/\underline{b}) = 0$). The result then follows (3). ■

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

Definition 4.7 — (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:

1. 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 4.25 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.

Proof. If $\mathcal{S} = \emptyset$, the result holds vacuously. Otherwise $\mathcal{S}_0 = \{s_0\}$. Let $s_1, s_2 \in \mathcal{S}$ and $\underline{a}, \underline{a}' \in \mathcal{A}ct^*$ such that $\underline{a} \sim^p \underline{a}'$, and assume $s_0 \xrightarrow{\underline{a}} s_1$ and $s_0 \xrightarrow{\underline{a}'} s_2$. Since LC is confluent, $s' \in \mathcal{S}$ exists such that $s_1 \xrightarrow{\underline{a}'/\underline{a}} s'$ and $s_2 \xrightarrow{\underline{a}/\underline{a}'} s'$. Since $\underline{a} \sim^p \underline{a}'$ we have by lemma 4.24(2) that $\underline{a}'/\underline{a} = \epsilon$ and $\underline{a}/\underline{a}' = \epsilon$. Therefore $s_1 = s_2$. ■

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.

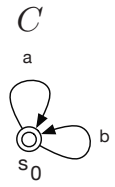


Figure 4.14: *A confluent constraint is not necessarily np-repulsing.*

Example 4.19 Consider constraint C depicted in Figure 4.14. C is confluent and p-attracting (consistent with lemma 4.25). 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 is also holds for constraints that are non-recursive. This is proven in the following lemma.

Lemma 4.26 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.

Proof. If $\mathcal{S} = \emptyset$, the result holds vacuously. Otherwise $\mathcal{S}_0 = \{s_0\}$. Let $s \in \mathcal{S}$ and $\underline{a}, \underline{a}' \in \mathcal{A}ct^*$, and assume $s_0 \xrightarrow{\underline{a}} s$ and $s_0 \xrightarrow{\underline{a}'} s$. We have to show that $\underline{a} \sim^p \underline{a}'$. Since C is confluent, $s' \in \mathcal{S}$ exists such that $s \xrightarrow{\underline{a}'/\underline{a}} s'$ and $s \xrightarrow{\underline{a}/\underline{a}'} s'$. Applying the property of confluence again we know that $s'' \in \mathcal{S}$ exists such that $s' \xrightarrow{(\underline{a}/\underline{a}')/(\underline{a}'/\underline{a})} s''$ and $s' \xrightarrow{(\underline{a}'/\underline{a})/(\underline{a}/\underline{a}')} s''$. But then by lemma 4.24(4) we know that $s' \xrightarrow{\underline{a}/\underline{a}'} s''$ and $s' \xrightarrow{\underline{a}'/\underline{a}} s''$. This pattern can be applied repeatedly. Then if $\underline{a}'/\underline{a} \neq \epsilon$ or $\underline{a}/\underline{a}' \neq \epsilon$, this would imply C to be either a recursive automaton or an automaton with an infinitely many states. Therefore we must have that $\underline{a}'/\underline{a} = \epsilon$ and $\underline{a}/\underline{a}' = \epsilon$. By lemma 4.24(2) we then know that $\underline{a} \sim^p \underline{a}'$. ■

From Lemma's 4.25 and 4.26 we thus know that every confluent logistics automaton and non-recursive constraint is both np-repulsing and p-attracting. The other way around is not true, which is demonstrated in the following example.

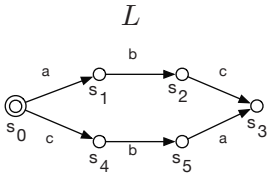


Figure 4.15: *An np-repulsing, p-attracting logistics automaton is not necessarily confluent.*

Example 4.20 Consider automaton L depicted in Figure 4.15. 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 4.23 to apply, the logistics automaton should be np-repulsing and the constraint should be p-attracting. From Lemma's 4.25 and 4.26 we thus know this holds in case of a confluent automaton and constraint.

4.6 Exploiting the Algebra

In this chapter 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 4.21 Assume that we want to compare specifications $(((L_1 \odot L_2) \upharpoonright C_1) \odot L_3) \upharpoonright C_2$ and $((L_1 \odot L_2) \upharpoonright C_1) \odot L_3$ 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 $(((L_1 \odot L_2) \upharpoonright C_1) \odot L_3) \upharpoonright C_2$ as follows:

$$
\begin{aligned}
&(((L_1 \odot L_2) \upharpoonright C_1) \odot L_3) \upharpoonright C_2 \\
&\approx \{\text{Distributivity of } \upharpoonright \text{ (Lemma 4.15)}\} \\
&\big((L_1 \odot L_2) \upharpoonright C_1 \upharpoonright C_2\big) \odot (L_3 \upharpoonright C_2) \\
&\approx \{\text{Commutativity of } \upharpoonright \text{ (Lemma 4.14),} \\
&\text{substitutivity of } \approx \text{ under } \odot \text{ (Lemma 4.12)}\} \\
&\big((L_1 \odot L_2) \upharpoonright C_2 \upharpoonright C_1\big) \odot (L_3 \upharpoonright C_2) \\
&\approx \{\text{Distributivity of } \upharpoonright \text{ (Lemma 4.15),} \\
&\text{substitutivity of } \approx \text{ under } \upharpoonright \text{ (Lemma 4.16)} \\
&\text{and substitutivity of } \approx \text{ under } \odot \text{ (Lemma 4.12)}\} \\
&\big((L_1 \upharpoonright C_2 \odot L_2 \upharpoonright C_2) \upharpoonright C_1\big) \odot (L_3 \upharpoonright C_2)
\end{aligned}
$$

$\sqsubseteq \{L_1 \restriction C_2 \sqsubseteq L_1, L_2 \restriction C_2 \sqsubseteq L_2, L_3 \restriction C_2 \sqsubseteq L_3$ by Theorem 4.23, substitutivity of $\sqsubseteq$ under $\restriction$ (Lemma 4.16) and $\odot$ (Lemma 4.12) and commutativity of $\odot$ (Lemma 4.11)$\}$
$((L_1 \odot L_2) \restriction C_1) \odot L_3$

Hence by Lemma 4.5 $\mathcal{L}((((L_1 \odot L_2) \restriction C_1) \odot L_3) \restriction C_2) \subseteq \mathcal{L}(((L_1 \odot L_2) \restriction C_1) \odot L_3)$. Further by Lemma 4.4 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 Lemmas 4.9 and 4.4 the optimization-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 $((L_1 \odot L_2) \restriction C_1) \odot L_3$. Therefore a solution to the BMO problem of $\big(((L_1 \odot L_2) \restriction C_1\big) \odot L_3) \restriction C_2$ is also a valid activity sequence of $((L_1 \odot L_2) \restriction C_1) \odot L_3$ (Lemma 3.1) establishing a makespan bound to any solution to the BMO problem of $((L_1 \odot L_2) \restriction C_1) \odot L_3$ (Lemma 3.1). This bound is actually a suboptimal solution of the BMO problem of $((L_1 \odot L_2) \restriction C_1) \odot L_3$ and is easier to compute than the optimal solution thereof.

Example 4.22 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 apply a p-attracting constraint C_3 on L_2.

First we try to find a p-attracting constraint C_3 on L_2. Then by Theorem 4.23 we know that $L_2 \restriction C_3 \sqsubseteq L_2$.

Subsequently by applying substitutivity of $\sqsubseteq$ under $\odot$ (Lemma 4.12) 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 $\restriction$ (Lemma 4.16), 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 4.21 we can now compute the BMO solution to $(L_1 \restriction C_1 \odot L_2 \restriction C_3) \restriction C_2$ which establishes a bound to the BMO solution of $(L_1 \restriction C_1 \odot L_2) \restriction C_2$.

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 a np-repulsing logistics automaton. Notice 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 is np-repulsing and for which $\mathcal{L}(L) = \mathcal{L}(\text{Tree}(L))$.
2. The logistics automaton describing a batch is then obtained via the composition of all individual product automata. By Lemma 4.20 we know the resulting automaton is also np-repulsing.
3. Each system constraint should preferably be written as a p-attracting constraint automaton. This is important since we know, by Lemma 4.4 and Theorem 4.23, that 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 some system constraints cannot be written (or conveniently written) as p-attracting automata step 4 (applying heuristics) is still applicable. This is because the constraining of a logistics automaton with any constraint preserves its np-repulsiveness (Lemma 4.21).
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 constraint automata to ensure that the optimization-space is effectively pruned, following Theorem 4.23.

This method allows a systematic approach to the specification of batch-oriented logistics which keeps 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).

We demonstrate the applicability and effectiveness of the algebra and the specification method in the next section by applying it to solve the BMO problem of the Buffered Twilight system for different batch sizes.

4.7 Optimizing the Buffered Twilight

To illustrate the approach introduced in this chapter we apply it to the Buffered Twilight system introduced in Section 4.1.1. The full specification of its plant and activities can be found in Appendix B. We start by giving a more detailed description of the system, its plant and activities. Table 4.3 depicts the decomposition of the Buffered Twilight system into resources and peripherals. Note that in comparison with the regular Twilight system the only difference is the inclusion of the Buffer resource (BUFFER). This resource only has one clamp peripheral BUFFER.CL.

Table 4.4 list all the activities of the Buffered Twilight. Notice that activities **1** to **12** are the same as those of the Twilight system (see Table 2.3 of Section 2.3.3) but activities **13** to **17** are specific to the Buffered Twilight. These include activities **13** to **16** (LR_PickFromBuffer, UR_PickFromBuffer, LR_PutOnBuffer and UR_PutOnBuffer) which capture the picking and placing of a product from/on the Buffer resource by the LR and UR resources. Further activity **17** (DrillFast) captures the second manufacturing operation of the DRILL resource for the *double pass* product flow. The complete specification of the activities of the Buffered Twilight can be found in Appendix B.

4.7.1 Regular and Double-pass Product flows

The Buffered Twilight system allows for the manufacturing of two types of products: *regular* products and *double-pass* products. We assume that each product i in the batch of n products ($1 \leq i \leq n$) is associated with a copy of the Buffered Twilight activities listed in Table 4.4. This is indicated by aliases **activity**.i. Figure 4.16 depicts the logistics automata L_{rp} capturing the *regular* product flow and L_{dpp} capturing the *double pass* product flow.

Product flow L_{rp} is similar to the one described in Section 2.1.3 for the Twilight system, except that it also accounts for the possibility of placing/picking a product on/from the Buffer resource after the input of a product (activity **1**.i), conditioning of a product (activity **11**.i) and drilling of a product (activity **12**.i).

Recall that the LR and UR robots have pre-defined locations and ranges (Section 2.1.4). Both the LR and UR resources are able to reach the BUFFER resource. However, since only the LR resource is able to pick a product from the IN resource, only the LR resource can place the product on the BUFFER after picking it from the IN resource. In the same way, only the UR resource is able to place a product on the OUT resource, and therefore only the UR resource is

Table 4.3: *Buffered Twilight resources and peripherals.*

Resource	**Peripherals**
(LR) Load Robot	LR.Z, LR.R and LR.CL
(UR) Unload Robot	UR.Z, UR.R and UR.CL
(DRILL) Drill	DRILL.Z, DRILL.D and DRILL.CL
(COND) Conditioner	COND.H, COND.A and COND.CL
(BUFFER) Buffer	BUFFER.CL
(CA) Collision Area	-
(IN) Input buffer	-
(OUT) Output buffer	-

Table 4.4: *Set of activities of the Twilight Buffered system.*

(**1**) LR_PickFromInput	(**5**) LR_PutOnDrill	(**9**) UR_PutOnCond
(**2**) LR_PutOnCond	(**6**) UR_PickFromDrill	(**10**) UR_PutOnOutput
(**3**) LR_PickFromCond	(**7**) UR_PickFromCond	(**11**) Condition
(**4**) LR_PickFromDrill	(**8**) UR_PutOnDrill	(**12**) Drill
(**13**) LR_PickFromBuffer	(**14**) LR_PutOnBuffer	(**15**) UR_PickFromBuffer
(**16**) UR_PutOnBuffer	(**17**) DrillFast	

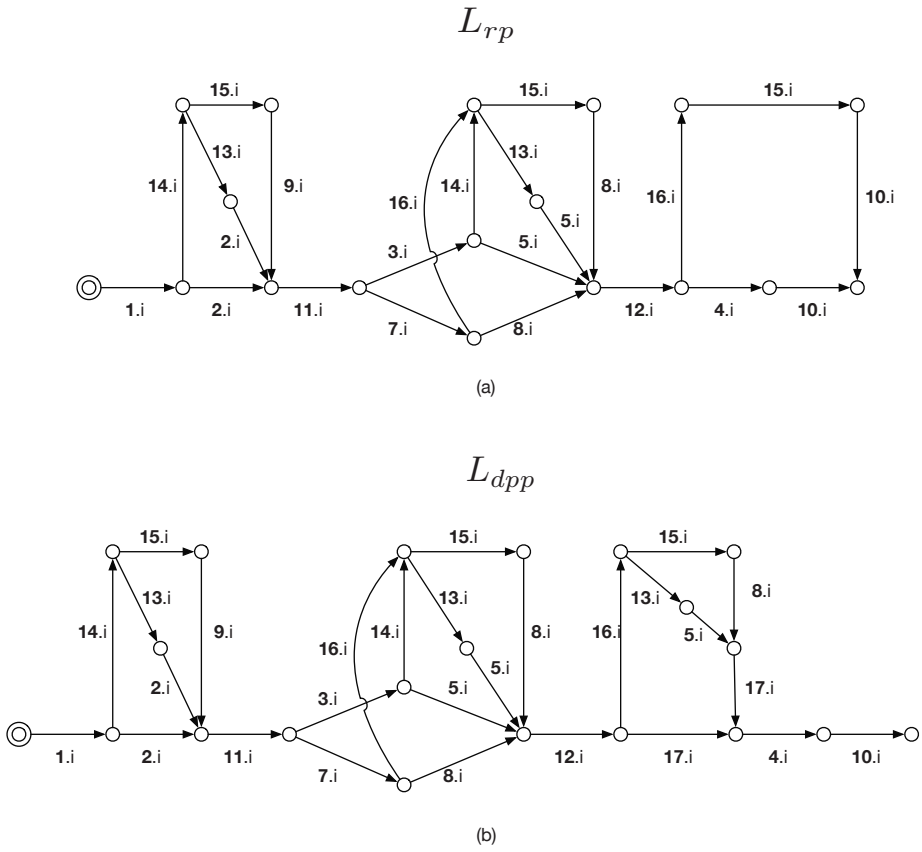


Figure 4.16: *Logistics automata capturing the product flow of a* regular *product (L_{rp}) and* double-pass *product (L_{dpp}).*

able to move a product from the BUFFER to the OUT resource. This is depicted in Figure 4.16 (a) by the activity sequence **15**.i **10**.i at the end of the product flow. For the other two moments, after Conditioning (activity **11**.i) or Drilling (activity **12**.i), both the LR and UR resources are able to pick/place products from/on the Buffer resource.

Now consider automaton L_{dpp} depicted in Figure 4.16 (b) which captures the *double-pass* product flow. In this flow a product must undergo two manufacturing activities on the DRILL resource, first activity Drill (**12**.i) and after that activity DrillFast (**17**.i). To this end the picking/placing of a product from/on the BUFFER resource differs from the *regular* product flow only after the Drill

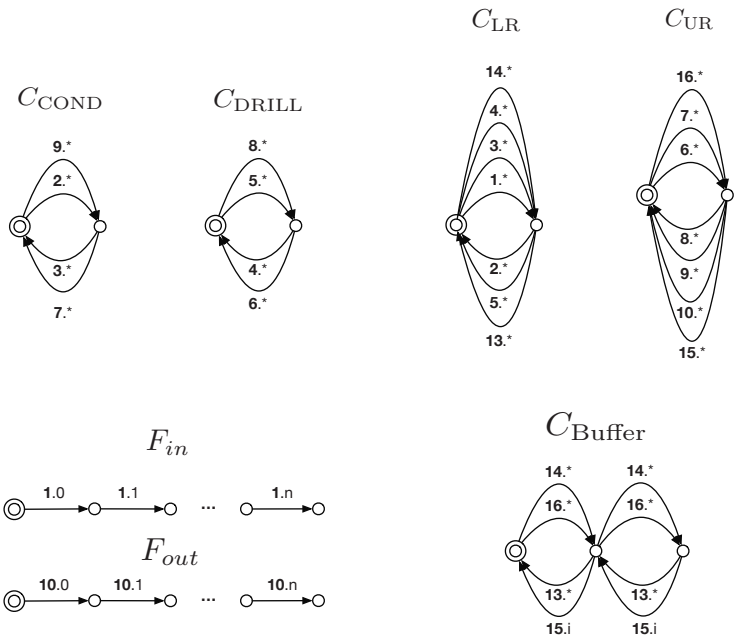


Figure 4.17: *Constraint automata C_{COND}, C_{LR}, C_{UR}, C_{DRILL} and C_{BUFFER} capturing capacity constraints and F_{in} and F_{out} capturing the product input ordering.*

activity. Immediately after Drill activity, two choices can be made. Either the UR resource places the product on the BUFFER resource (activity **16**.i) or the DRILL resource immediately executes the DrillFast activity (activity **17**.i). If the product is placed on the BUFFER resource, both the UR and LR resources can then return the product to the DRILL to perform the DrillFast activity (activity sequences **13**.i **5**.i **17**.i and **15**.i **8**.i **17**.i).

For readability reasons L_{rp} and L_{dpp} are not written as np-repulsing automata. It is easier to write them in this form before they are used for optimization purposes. In fact we have done so in the optimization steps that follow.

4.7.2 System Constraints

Figure 4.17 depicts the capacity and input ordering constraints of the Buffered Twilight. Constraint automata C_{COND}, C_{DRILL}, C_{LR}, C_{UR}, and C_{BUFFER} capture the capacity constraints of all resources of the Buffered Twilight. To avoid cluttering the automata figures we write **a**.$*$ to represent n different transi-

tions with labels $\mathbf{a}.1, \cdots, \mathbf{a}.n$ (where **a** denotes the name of an activity). The BUFFER resource has a capacity of two products and is therefore captured by a three-state constraint automaton C_{BUFFER}. The initial state represents that the BUFFER is empty, the middle state that the resource is occupied by one product and the final state that the BUFFER is full. Compared to the Twilight system C_{LR} and C_{UR} contain transitions to include the picking and placing of products from/on the BUFFER resource (activities **13**.i, **14**.i, **15**.i and **16**.i). Finally, the FIFO end-to-end ordering of products is capture by automata F_{in} and F_{out}. Notice that these capacity constraints are all p-attracting automata implying that their application effectively reduces state-spaces and optimization-spaces.

4.7.3 Results

Now that we have introduce the Buffered Twilight, its logistics requirements and constraints we compute the optimization-space and the solution to the BMO for batch sizes between 2 to 20 products. The values presented in this section are computed using 8 Intel i7 920@2.67Ghz CPUs with 32GB of memory. We use the CIF tool [18] to compute the state-spaces of logistics automata and the algorithms developed in the previous chapter to compute the corresponding optimization-spaces and optimal activity sequences.

Table 4.5 lists the state-space and optimization-space sizes together with the optimal makespan for different batch sizes. For brevity we will let L_{bf_n} denote the logistics automaton of the Buffered Twilight for a batch of n products. L_{bf_n} is defined by: $(L_{rp_1} \odot L_{dpp_2} \odot \cdots \odot L_{rp_{n-1}} \odot L_{dpp_n}) \upharpoonright F_{in} \upharpoonright F_{\text{out}} \upharpoonright C_{\text{COND}} \upharpoonright C_{\text{DRILL}} \upharpoonright C_{\text{LR}} \upharpoonright C_{\text{UR}} \upharpoonright C_{\text{BUFFER}}$. Note that logistics expression $(L_{rp_1} \odot L_{dpp_2} \odot \cdots \odot L_{rp_{n-1}} \odot L_{dpp_n})$ specifies an alternating sequence of *regular* and *double-pass* product flows. Further notice that if we write L_{rp} and L_{dpp} as np-repulsing automata (which can always be done) then by Lemma 4.20 the complete logistics expression is np-repulsing as well (which is required for our main reduction Theorem 4.23 to apply).

Clearly, for the Buffered Twilight we are only able to compute the optimal activity sequences for batches of 2, 3 and 4 products. The optimization-space grows extremely fast in the size of the batch; from 2 products to 4 products the state-space size increases with 838%. To address this we will rely on over-specification by subsequently introducing constraint automata C_{ov1_i} and C_{ov2} depicted in Figure 4.18.

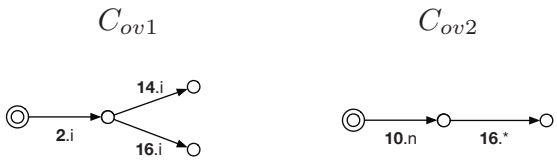


Figure 4.18: *Constraint automata C_{ov1_i} and C_{ov2} capturing two over-specification requirements.*

Over-specification: Restriction the access to the Buffer

In our specification of the Buffered Twilight we allow the system to use the BUFFER resource in many different phases of the product flow. By analyzing the solutions to the BMO problem for the original specification (for the smaller batches sizes) we observed that a product is never placed on the BUFFER immediately after input. Therefore, we figured that a proper heuristic is defined by constraint automaton C_{ov1_i} (depicted in Figure 4.18) that only allows a product i (with $1 \leq i \leq n$) to be placed on the buffer after it has been placed on the COND resource (and thus eliminating the possibility of using the BUFFER resource after the product is inputted). We apply these constraints to obtain: $L_{bf_n} \restriction C_{ov1_1} \restriction \cdots \restriction C_{ov1_n}$. Notice C_{ov1_1} to C_{ov1_n} are p-attracting automata. Since L_{bf_n} is np-repulsing we know by Theorem 4.23 that the constraining will reduce the optimization-space.

This is also confirmed by the experiments. The results of which are shown in Table 4.6. Notice that the addition of C_{ov1} allows us to compute the optimal makespan activity sequence for a batch of 6 products (which we could only do for 4 products in the original specification). Furthermore, the solutions for the BMO problem of $L_{bf_n} \restriction C_{ov1_1} \cdots \restriction C_{ov1_n}$ are also optimal solutions to the BMO of L_{bf_n}. In other words, the of C_{ov1} did not results in sub-optimal solutions to the BMO of L_{bf_n}. Since we are not able to compute solutions for batches larger than 6 products, we will add another non-essential constraint to the system in the form of C_{ov2} depicted in Figure 4.18.

Over-specification: Removing assignment choices

Again by analyzing the solutions to the BMO problem for the original specification (for the smaller batches sizes) we also observed that picking/placing of product from/on the BUFFER resource is done mostly by the LR resource. Therefore, we figured that another useful heuristic is defined by constraint automata C_{ov2} which captures the assignment of the LR resource as the only

Table 4.5: *Size of the state-space of the L_{bf_n} and MaxPlus(L_{bf_n}), and computed makespan for different batch sizes.*

	State-Space		Optimization-Space		
Batch Size	N. States	N. Edges	N. States	N. Edges	Makespan
2	174	353	5227	8528	62.6
3	1030	2468	180030	132074	94.0
4	5700	13872	2344026	3960469	116.6
5	14475	35020	-	-	-
6	23463	57093	-	-	-
7	-	-	-	-	-
8	-	-	-	-	-
9	-	-	-	-	-
10	-	-	-	-	-

Table 4.6: *State-space sizes of $L_{bf_n} \restriction C_{ov1_1} \cdots \restriction C_{ov1_n}$ and MaxPlus($L_{bf_n} \restriction C_{ov1_1} \restriction \cdots \restriction C_{ov1_n}$, and computed makespan for different batch sizes.*

	State-Space		Optimization-Space		
Batch Size	N. States	N. Edges	N. States	N. Edges	Makespan
2	160	353	1921	2833	62.6 (0%)
3	779	1656	25242	42943	94.0 (0%)
4	2803	6365	163652	319135	116.6 (0%)
5	7594	17881	545867	1129530	145.5
6	17166	40660	1384970	2920355	170.6
7	36450	87100	-	-	-
8	74738	178216	-	-	-
9	151874	363976	-	-	-
10	-	-	-	-	-

Table 4.7: *State-space sizes of $L_{bf_n} \restriction C_{ov1_1} \cdots \restriction C_{ov1_n} \restriction C_{ov2}$ and MaxPlus($L_{bf_n} \restriction C_{ov1_1} \restriction \cdots \restriction C_{ov1_n} \restriction C_{ov2}$), and computed makespan for different batch sizes.*

	State-Space		Optimization-Space		
Batch Size	N. States	N. Edges	N. States	N. Edges	Makespan
2	120	190	598	877	64.4 (-3.0%)
3	459	842	4355	7648	95.0 (-1.0%)
4	1286	2569	19704	37608	121.5 (-4.2%)
5	2992	6135	57585	113853	151.5
6	6420	13387	130561	259335	178.5
7	13244	27651	282085	564315	207.9
8	26956	56659	573989	1146243	234.9
9	54252	113715	1180085	2366163	264.5
10	109100	229747	-	-	-

resource able to reach the BUFFER resource. This is enforced by specifying that activities related to the picking and placing of products from/on the BUFFER resource by the UR resource can only occur after the last product of the batch has been outputted. Essentially this means that these activities are never executed, since they can only executed once all product automata have reached their final states and then these activities are blocked due to synchronization. By applying this constraint we obtain $L_{bf_n} \restriction C_{ov1_1} \cdots \restriction C_{ov1_n} \restriction C_{ov2}$. Since C_{ov2} is a p-attracting automaton its application will reduce the optimization-space.

The results of state-space and optimization-space sizes for $L_{bf_n} \restriction C_{ov1_1} \cdots \restriction C_{ov1_n} \restriction C_{ov2}$ together with the makespan are shown in Table 4.7. By adding constraint C_{ov2} we have managed to compute the optimization-space up to 9 products in the batch. This comes at the cost of finding sub-optimal solutions for batches of 2 products (a loss of 3.0%) and 4 products (a loss of 4.2%). These results establish a sub-optimal solution of the BMO of the original unconstrained Buffered Twilight system as well as a conservative bound on the makespan. The relative difference in the computed makespan is shown between parenthesis next to the makespan values.

4.8 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 [24] and of Supervisory Control Theory (SCT) [66] developed in [70] and applied in [19], [36], [80]. 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) [59], 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 [5] or vehicle routing [22] problems. In addition to our approach, some of these solutions include deadlines and due dates [62], [63], [84], [85] and other types of timing constraints [6], [7]. However, they do not focus on expressing or guaranteeing the satisfaction of functional requirements as discussed in Chapter 2. 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 [5] 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).

4.9 Conclusions

In this chapter we defined equivalence and inclusion relations on logistics automata and their languages. 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 and that the later is always included in the corresponding Tree automaton encoding the worst-case optimization-space. We proved that the equivalence and inclusion relations are substitutive under the composition, constraining, (max,+) and Tree operators. In addition, we proved commutativity, associativity and distributivity properties for the different operators. In summary, the composition operator is both commutative and associative and the constraining operator is commutative and distributive over the composition operator. With these ingredients we defined 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.

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. 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 the Buffered Twilight system for different batch sizes. We observed that by specifying the system requirements alone we were able to compute solutions to the BMO problem for batches of 4 products. However, by applying our heuristic approach, where we systematically add constraints to the specification, we were able to obtain solutions for batches of up to 9 products.

5 | Bottleneck Identification using Stochastic Criticality Analysis

The goal of the framework presented in this thesis is the design exploration and makespan optimization of flexible manufacturing systems. This includes exploring different types of resources and peripherals and different system layouts. For instance geometrical location of resources and number of resources and routing options for the product flow.

In the previous chapters we discussed the specification and optimization of flexible manufacturing systems by computing their makespan optimal activity sequence for a batch of products. In this chapter we discuss the identification of performance bottlenecks of activity sequences as a means to identify possible improvement candidates for a new design iteration. This chapter is based on the work in [16]. In this context, the identification of performance bottlenecks is fundamental, since no other modifications in the system design will lead to an improvement in performance [52]. To this end we could rely on well-known methods such as the Critical-Path Method (CPM) and Program Evaluation and Review Technique (PERT). These methods rely on the identification of critical paths and critical actions of activities. The CPM is concerned with fixed-execution times and the PERT can deal with stochastic execution times. However, recent research has shown that more informative feedback is obtained by the study of the criticality of actions [32], instead of the criticality of paths as is done in classical CPM and PERT approaches.

In this chapter we consider the use of criticality of actions as an analysis technique for bottleneck identification in flexible manufacturing systems. To this end we extend our framework with stochastic execution times of actions. There can be production activities which operate with slower dynamics (for

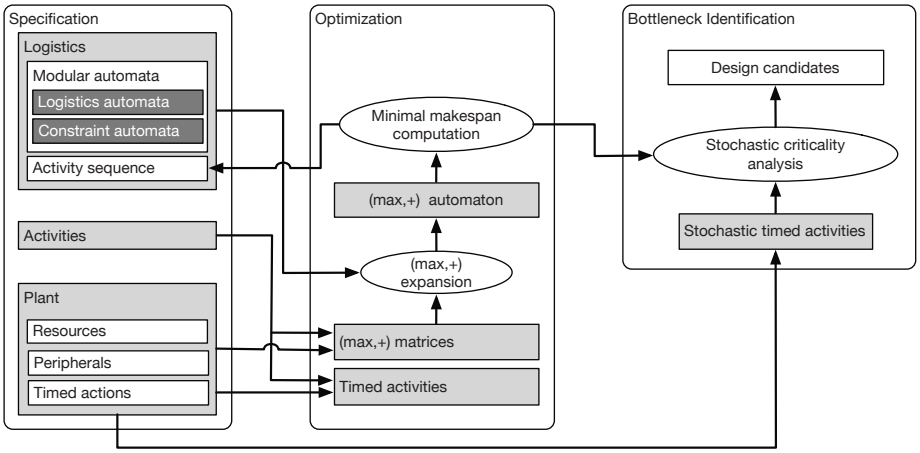


Figure 5.1: *Overview of the framework concepts including the* Bottleneck Identification *domain.*

instance the process of heating a product in the Twilight system) or actions whose execution times depended on initial conditions (for instance the process of alignment in the Twilight system). The temporal behavior of such actions is better captured by assuming variable/stochastic execution times instead of fixed execution times.

Figure 5.1 depicts the overview of our framework now including the *bottleneck identification* domain which we discuss in this chapter. Given a system specification the optimization approach (discussed in Chapters 3 and 4) determines the makespan optimal activity sequence. This activity sequence is then analyzed to identify performance bottlenecks in order to find candidates to improve makespan in the next design iteration. This requires that we consider the start and end times of nodes of an activity. Since we cannot abstract from the timing of the individual nodes (or actions) we cannot rely on their (max,+) characterization introduced in Chapter 3. Furthermore, we want to capture system actions that exhibit variations in their executions times. Therefore, in this chapter we define the notions of *stochastic timed actions and activities* and introduce a *stochastic criticality analysis* technique as a method to identify performance bottlenecks when peripheral actions can exhibit both fixed and stochastic execution times. Finally we use our Twilight system as a running example to illustrate our bottleneck identification approach.

This chapter is organized as follows. Section 5.1 defines stochastically timed actions and activities. Section 5.2 introduces the notions of critical path and critical node. Section 5.3 presents the stochastic criticality analysis in detail. Section 5.4 discusses the interpretation and visualization of the results of the stochastic criticality analysis. Section 5.5 uses the Twilight system as an application example to illustrate the analysis technique. Section 5.6 discusses the related work and Section 5.7 concludes the chapter.

5.1 Stochastically Timed Actions and Activities

In this section we introduce the notions of *stochastically timed actions* and *stochastically timed activities*. The later notion is based on the redefinition of the *start* and *end* times of nodes in an activity for actions with stochastic execution times. These concepts will then be used to compute the *critical path* and *critical nodes* of a stochastically timed activity.

To model the timing variations that a system exhibits, we attribute to each action a random variable representing the random execution time of that action. To this end we assume $\mathcal{T}$ to denote the collection of all such random variables. We assume that this set contains random variable $\underline{0}$ which takes value 0 with probability 1.We start by defining the stochastic execution time of an action and of a node.

Definition 5.1 — (Stochastically timed action). We assume a function $E : \mathcal{A} \to \mathcal{T}$ that maps each action to its corresponding random execution time variable.

Definition 5.2 — (Stochastic execution time of a node). Given activity $(N, \to)$ and node $n \in N$, we define the execution time of node n as:

$$E(n) = \begin{cases} E(x) & \text{if } M(n) = (x, p) \\ & \text{for some } x \in \mathcal{A}, p \in \mathcal{P} \\ \underline{0} & \text{otherwise.} \end{cases}$$

Further we need to adapt the computation of the start and end times of actions assuming their stochastic execution. This is done in following definition.

Definition 5.3 — (Stochastic start and completion time of a node). Given activity $a = (N, \rightarrow)$ we define the random variables start time $S(n)$ and finishing time $F(n)$ for each node $n \in N$:

$$S(n) = \begin{cases} \underline{0} & \text{if } M(n) = (r, cl) \\ & \text{for some } r \in \mathcal{R} \\ \max\limits_{n' \in Pred(n)} F(n') & \text{otherwise} \end{cases}$$

$$F(n) = S(n) + E(n)$$

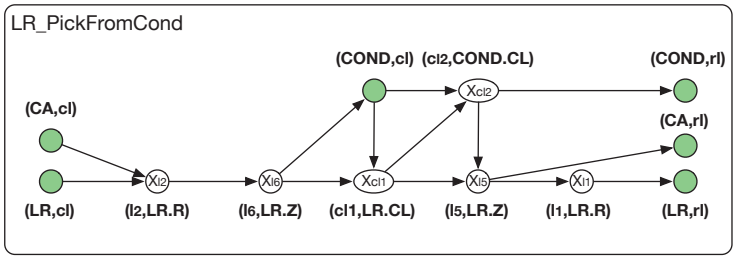


Figure 5.2: *Activity LR_PickFromCond assuming stochastic execution times for actions. These are captured by the random variables $X_{l2}, X_{l6}, X_{cl1}, X_{cl2}, X_{l5}$ and X_{l1}.*

Example 5.1 Consider activity LR_PickFromCond in Figure 5.2 assuming stochastic execution times for peripheral actions. Note that instead of fixed execution times, the temporal behavior of actions are now defined by random variables X_{l2}, X_{l6}, X_{cl1}, X_{cl2}, X_{l5} and X_{l1}. Each random variable is associated with a discrete or continuous distribution. To capture fixed execution times we assume the random variable to take this execution time with probability one.

5.2 Critical Path and Critical Node

In the previous section we defined stochastically timed actions and activities. Using these concepts we define in this section the concepts of *critical node* and *critical path* which we need for the Stochastic Criticality Analysis. We start by defining paths and makespans of a paths.

Definition 5.4 — (Path and Makespan of a Path). Given an activity $a = (N, \rightarrow)$ we let Δ denote the set of all paths of nodes $\underline{n} = n_1 \cdots n_k$ such that $n_i \rightarrow n_{i+1}$ for each $i : (1 \leq i < k)$, $\text{Pred}(n_1) = \emptyset$ and $n_k \not\rightarrow$. For each path $\underline{n} = n_1 \cdots n_k \in \Delta$, we define the makespan as $mks(\underline{n}) = F(n_k)$. Notice that $mks(\underline{n})$ is a random variable.

The makespan of a path represents the total time necessary to complete the execution of all peripheral actions in that path. A path for which the makespan is larger than or equal to the makespan of any other path in Δ and for which the start time of every node is the same as the finishing time of its predecessor is called a *critical path*.

Definition 5.5 — (Critical Path). For each $\underline{n} = n_1 \cdots n_k \in \Delta$, we define a random variable:

$$C(\underline{n}) = \begin{cases} 1 & \text{if } mks(\underline{n}) = max_{\underline{n'} \in \Delta} mks(\underline{n'}) \\ & \text{and } S(n_{i+1}) = F(n_i) \text{ for all } 1 \leq i < k-1 \\ 0 & \text{otherwise} \end{cases}$$

Hence for each path $\underline{n} \in \Delta$, indicator $C(\underline{n})$ indicates whether $\underline{n}$ is a critical path or not. Notice that $C(\underline{n})$ is a Bernoulli distributed random variable. Note that there can be multiple paths which are critical. All these critical paths must then have the same makespan.

Definition 5.6 — (Critical Node). For each $n \in N$, we define a random variable:

$$C(n) = \begin{cases} 1 & \text{if for some path } \underline{n} = n_1 \cdots n_k \in \Delta \text{ with } C(\underline{n}) = 1, n = n_i \\ & \text{for some } 1 \leq i \leq k \\ 0 & \text{otherwise} \end{cases}$$

Therefore, for each node $n \in N$, $C(n)$ indicates whether n is on a critical path or not. This node can be on multiple critical paths. Notice that $C(n)$ is a Bernoulli distributed variable.

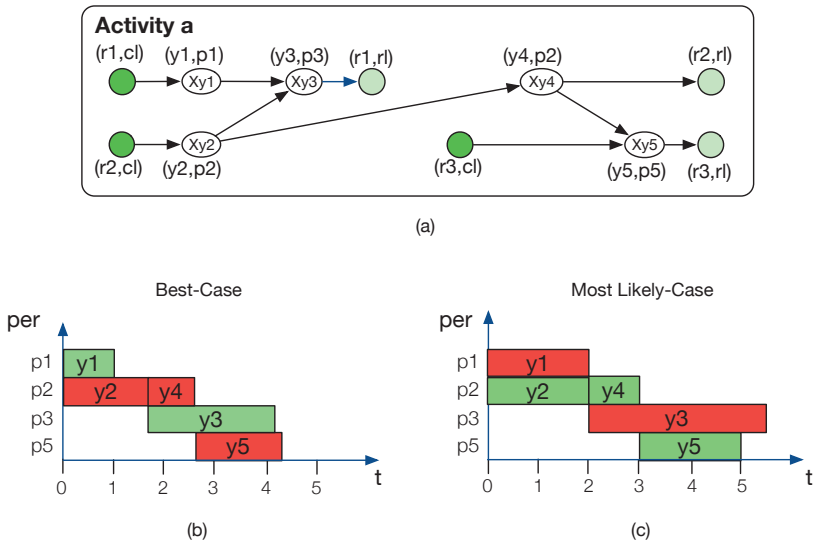


Figure 5.3: *Different execution scenarios of activity a for (b) the best-case execution times and (c) the most likely-case execution times.*

Table 5.1: *Execution times of the actions of activity a of Figure 5.2 for best, worst and most-likely case.*

	Execution time		
Action	Best-case	Worst-case	Most likely-case
y1	1	3	2
y2	1.8	2.2	2
y3	2.5	4	3.5
y4	0.8	1.2	1
y5	1.8	2.2	2

Example 5.2 Consider activity a depicted in Figure 5.3 (a). Activity a uses actions $y1, y2, y3, y4$ and $y5$ and peripherals p_1 and p_3 associated with resource r_1, p_2 associated with resource r_2 and p_5 associated with resource r_3. Table 5.1 list the executions times of each action assuming different cases. The *best-case* represents the minimum, the *worst-case* the maximum and the *most-likely* case the mode of the distribution associated with the random variables $X_{y1}, X_{y2}, X_{y3}, X_{y4}$ and X_{y5}. Figure 5.3 (b) depicts the Gantt chart of the execution of activity a assuming the best-case execution times for each action. Critical action nodes are colored in red. In this case the critical path is (r_1, cl) $(y2, p_2)$ $(y4, p_2)$ $(y5, p_5)$ (r_3, rl) and nodes (r_1, cl), $(y2, p_2)$, $(y4, p_2)$, $(y5, p_5)$ and (r_3, rl) are critical nodes.

Now consider the alternative most likely-case execution of activity a shown in Figure 5.3 (c). The corresponding execution times of each action are now associated with the most-likely execution times listed in Table 5.1. In this case the critical path is (r_1, cl) $(y1, p_1)$ $(y3, p_1)$ (rl, r_1) and (r_1, cl), $(y1, p_1)$, $(y3, p_1)$ and (rl, r_1) are critical nodes. Note that for different samples from the distribution associated with the random variables $X_{y1}, X_{y2}, X_{y3}, X_{y4}$ and X_{y5} we obtain different critical nodes and critical paths. The node criticality for the best-case, worst-case and most-likely case execution times are summarized in Table 5.2. Observe that dependent on the type of execution time taken, different nodes are labeled critical. We omit the resource claim and release nodes in Table 5.2 since we assume their execution times to be negligible.

5.3 Stochastic Criticality Analysis

In this section we introduce the *Stochastic Criticality Analysis* (SCA) approach to identify performance bottlenecks in activities when actions can exhibit stochastic execution times. The SCA approach uses the *Criticality Index* (CI) [32], [83] metric instead of the typical *criticality of paths* used by the Critical-Path Method (CPM) (for fixed execution times) and Program Evaluation and Review Technique (PERT) (for stochastic execution times). The Criticality Index of a node represents the probability of that node occurring in a critical path. To estimate the CI value for each node of an activity the SCA approach relies on a Monte-Carlo long-run average simulation approach (including error estimation based on confidence intervals). Intuitively, the SCA approach will run k simulations, where each simulation is summarized in the following steps:

1. The execution time of each action in the activity is obtained by taking a sample of the distributions associated with the corresponding random variables of that action.
2. The critical paths of the activity with respect to the set of execution time samples are computed.
3. The critical nodes are computed and are given a criticality of 1. The remaining nodes obtain a criticality of 0.
4. The Criticality Index of each node is determined by averaging the criticalities of that node over all simulations runs.

The simulation terminates once the absolute estimation error of each node in the activity is below a certain bound. In the remainder of this section we formally define *criticality index* and its estimation, and explain how to use confidence intervals to determine the number of simulation runs required to obtain accurate estimations.

5.3.1 Criticality Index Estimation

We start by defining *Criticality Index* as the expected number of occurrences of a node being on a critical path of an activity.

Definition 5.7 — (Criticality Index). Given an activity $(N, \rightarrow)$, the Criticality Index $c(n)$ of a node $n \in N$ is the expected number of times that this node is on a critical path. This index is defined as $c(n) = E(C(n))$.

In order to estimate the criticality index $c(n)$ of a node n we rely on the calculation of confidence intervals. For notational simplicity, we fix n and write C to denote $C(n)$ and c to denote $c(n)$. Thus $c = \mathbb{E}(C)$. Since C has a Bernoulli distribution we know that the variance of C is given by $c(1-c)$.

To estimate c we define the point-estimator $\widehat{C} = \frac{1}{k}\sum_{i=1}^{k} C_i$, where each C_i represents an independent copy of C and where k represents the number of simulation runs. Then by the strong law of large numbers we know that $\widehat{C}$ converges strongly to c. Hence for a sufficiently large k, $\frac{1}{k}\sum_{i=1}^{k} c_i \approx c$, where each c_i represents a sample from C_i. We will write $\widehat{c}$ to denote point-estimation $\frac{1}{k}\sum_{i=1}^{k} c_i$.

Algorithm 3 Stochastic Critical Analysis algorithm

procedure StochasticCP(a, E, ϵ, z, k)
  $i = 1$
  for each node n in activity a **do**
    $c(n) = 0$
  while $error \geq \epsilon$ or $i \leq k$ **do**
    for each node n in activity a **do**
      $t(n) = sample(E_n)$
    $\underline{cp} =$ ComputeCriticalPaths(a, t)
    for each node n in $\underline{cp}$ **do**
      $c(n) = (c(n-1) \times (i-1) + 1)/i$
      $error(n) = (z \times \sqrt{c(n) \times (1 - c(n))})/\sqrt{i}$
      $error = max(error, error(n))$
    $i = i + 1$

5.3.2 Confidence Intervals and Determining the Number of Required Simulations

We would like to determine k such that the absolute error $|\, c - \widehat{c}\,|$ is sufficiently small. For this we use the central limit theorem stating that $\widehat{C} = \frac{1}{k}\sum_{i=1}^{k} C_i$ is approximately Normally distributed with expected value kc and variance $kc(1-c)$. Normalization yields $\frac{k\widehat{C}-kc}{\sqrt{kc(1-c)}}$ to be approximately $N(0,1)$ distributed. This also holds when c in the denominator is replaced by point-estimator $\widehat{C}$. Rewriting yields that $\sqrt{k}\frac{\widehat{C}-c}{\sqrt{\widehat{C}(1-\widehat{C})}}$ is approximately $N(0,1)$ distributed for sufficiently large k. Therefore $\mathbb{P}(-Z_{\frac{\gamma+1}{2}} \leq \sqrt{k}\frac{k\widehat{C}-kc}{\sqrt{\widehat{C}(1-\widehat{C})}} \leq Z_{\frac{\gamma+1}{2}}) \approx \gamma$ for confidence level γ and quantile $Z_{\frac{\gamma+1}{2}}$, where quantile $Z_p = \Phi^{-1}(p)$ and Φ is the cumulative distribution function of N(0,1). By rewriting this expression we obtain $\mathbb{P}(c \in [\widehat{C} - Z_{\frac{\gamma+1}{2}}\frac{\sqrt{\widehat{C}(1-\widehat{C})}}{\sqrt{k}}, \widehat{C} + Z_{\frac{\gamma+1}{2}}\frac{\sqrt{\widehat{C}(1-\widehat{C})}}{\sqrt{k}}]) \approx \gamma$. Replacing $\widehat{C}$ with point-estimation $\widehat{c}$ delivers confidence interval $[\widehat{c} - Z_{\frac{\gamma+1}{2}}\frac{\sqrt{\widehat{c}(1-\widehat{c})}}{\sqrt{k}}, \widehat{c} + Z_{\frac{\gamma+1}{2}}\frac{\sqrt{\widehat{c}(1-\widehat{c})}}{\sqrt{k}}]$ which contains c with confidence γ.

In case c is contained in this interval $|c - \widehat{c}| \leq Z_{\frac{\gamma+1}{2}}\frac{\sqrt{\widehat{c}(1-\widehat{c})}}{\sqrt{k}}$. We use this inequation to stop the simulation once the absolute error is smaller than a predetermined bound. In practice, for a number k of runs, samples $e_1(n), ..., e_k(n)$

from copies $E_1(n), E_2(n), ..., E_k(n)$ from $E(n)$ are taken, for every node $n \in N$. For each run m ($1 \leq m \leq k$) we use a longest path algorithm (see [47], [83]) to compute the critical path using samples $e_m(n)$ of each node. Then values $c_m(n)$ are computed for each node according to Definition 5.6, together with the current error bound. The simulation terminates once the bound of every node is smaller than or equal to a specified bound.

Algorithm 5.3 describes the implementation of the SCA technique. As input the analysis requires an activity a, stochastic execution time function E, error bound ϵ and quantile z corresponding to the desired confidence level. First, the CI of each node $c(n)$ is set to 0 after which we start a while loop. Within the scope of the loop, we first sample an execution time for each node n in the activity from the distribution associated with the corresponding random variable $X_n \in \mathcal{T}$. Using these samples we compute the critical paths $\underline{cp}$ and for each node n on the critical paths, $c(n)$ is updated. Thereafter we determined the maximum *error* bound which is the largest error bound computed for any of the nodes. The while loop is repeated while this *error* is larger than the desired error bound ϵ or timeouts after k runs of the loop. Once the loop terminates we have obtained the CI values for all the nodes in activity a.

5.4 Interpretation of the Criticality Index

In this section we provide an interpretation for the Criticality Index and discuss how it provides more information than classical critical path analyses. For this purpose we use as an example activity a of Figure 5.3 (a). We assume that the execution times of actions are modeled by random variables. Classical critical path analysis that consider stochastic execution times, such as the PERT approach [55], consider distinct cases for execution times, which correspond to the maximal, minimal and mode values of the distribution associated with the random variable. These are depicted as worst (*w*), best (*b*) and likely (*l*) cases in Table 5.1. The PERT approach first estimates the 'expected' time for each action, using equation $(b + 4 \times l + w)/6$, and then computes the critical paths with those values. The expected" execution times and the resulting criticality values (according to Definition 5.6) are depicted in Table 5.3, under column PERT. From these, we see that the critical nodes are $(y1, p_1)$ and $(y3, p_1)$, and therefore reducing the execution of these nodes will result in a lower makespan.

Table 5.2: *Node criticality for the different cases of execution times of Table 5.1*

	Criticality of nodes		
Node	Best-case	Worst-case	Most likely-case
$(y1, p_1)$	0	1	1
$(y2, p_2)$	1	0	0
$(y3, p_3)$	0	1	1
$(y4, p_2)$	1	0	0
$(y5, p_5)$	1	0	0

Table 5.3: *Comparison of the PERT and SCA approaches.*

	PERT		SCA
Node	Expected execution time	PERT node criticality	Criticality Indices (CI)
$(y1, p_1)$	2	1	0.52
$(y2, p_2)$	2	0	0.48
$(y3, p_3)$	3.4	1	0.72
$(y4, p_2)$	1	0	0.28
$(y5, p_5)$	2	0	0.28

Consider the same example and assumptions used for the PERT approach, but now applying the proposed SCA approach. We then obtain the Criticality Index (CI) of every node as defined in Definition 5.7. The results are depicted in Table 5.3, under column SCA. The stochastic analysis results indicates that, even though node $(y3, p_1)$ has the highest criticality, nodes $(y1, p_1)$ and $(y2, p_2)$ have an almost identical probability of occurring on a critical path. The fact that node $(y2, p_2)$ is also a potential candidate to improve is not exposed by the PERT analysis. The advantage of the stochastic criticality analysis is that it considers the total stochastic distribution together with the dependencies between nodes. This enables an enriched overview of the criticality of nodes which cannot be obtained by classical critical path approaches. This is further illustrated in the next section with the Twilight system case study.

5.5 Application to the Twilight System

In this section we compare the PERT and SCA approaches as a means to identify performance bottlenecks on the optimal activity sequence of the regular Twilight system for a batch of two products as obtained in Section 3.4. Since bottleneck improvement is always possible (there will always be a bottleneck), we insist that the (expensive) Drill resource should be the performance limiting resource.

5.5.1 Stochastic Time Modeling

To model the stochastic variability of the nodes in our example system, we model their timing behavior using PERT Distributions [1], [26]. The PERT distribution (also called beta-PERT) is a smooth version of the uniform distribution or triangular distribution. It is characterized by a minimal value, a maximal value, a mode and a shape parameter λ.

Table 5.4: *Set of distributions for our case study.*

Action	min	max	mode	λ
Clamp/Unclamp	0.2	0.4	0.25	8
LR Moves	5.0	6.2	6	8
UR Moves	4	6	5	8
Condition	4	16	11	5
Drill	4	26	10	5

The distributions for the actions are defined in Table 5.4. Regarding the variability in the system we assume that actions performed by all clamp peripherals and movement actions by the LR and UR resources exhibit low variance while the remaining actions of the peripherals belonging to the COND (Condition) and DRILL (Drill) resource exhibit high variance. The hight variation in the execution time of the Conditioning action is due to the differences in the initial temperatures of the inputted products. The variation exhibited by the Drill action is due to the assumption that depending on the desired depth and type of drilling profile each product may be confronted with different drilling times. As an application of the SCA method we will use the optimal activity sequence for the Twilight system with a batch of two products obtained in Section 3.4 (which we sequence using the ; operator): **1**.1;**2**.1;**1**.2;**11**.1;**7**.1;**2**.2;**8**.1;**12**.1;**11**.2;**3**.2;**6**.1;**5**.2;**10**.1;**12**.2;**6**.2;**10**.2.

5.5.2 Analysis and Results

We implemented both the proposed SCA analysis and the classical critical path analyses (CPM and PERT) within our framework. Visualization of the results is done using the TRACE viewer [42]. For this case study we assume a confidence level of 99% and an absolute error bound 0.001. To provide intuitive feedback we use colored Gantt charts.

Figures 5.4 and 5.5 depict the Gantt charts of the results of the PERT analysis and the SCA approach, respectively. In the case of the SCA approach the different shades of red of imply different criticality indices. The lighter the redness, the lower the criticality index, and vice-versa. We discuss in detail the results within the bounded area (denoted by the black lines in Figures 5.4 and 5.5).

Table 5.5: *Criticality obtained from PERT analysis for the aggregated actions of a peripheral.*

Actions p/ peripheral	B	W	L	PERT
COND.H	0	0	0	0
DRILL.D (1,2)	0	1	0	0
LR.R and LR.Z (1,2)	1	0	1	1
LR.R and LR.Z (3,4)	1	0	1	1
UR.R and LR.Z (1,2)	0	0	0	0

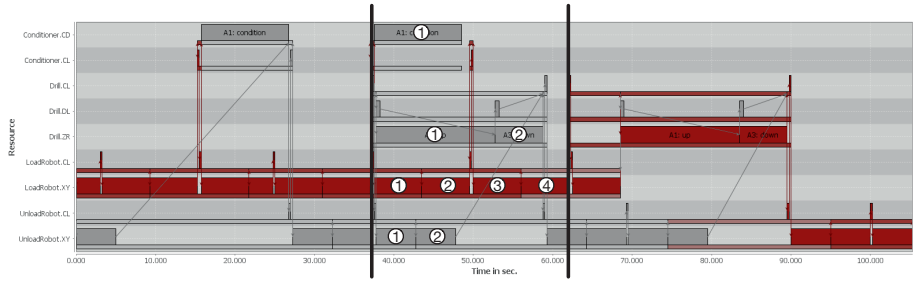


Figure 5.4: *Result of classical criticality analysis assuming expected action execution times (PERT).*

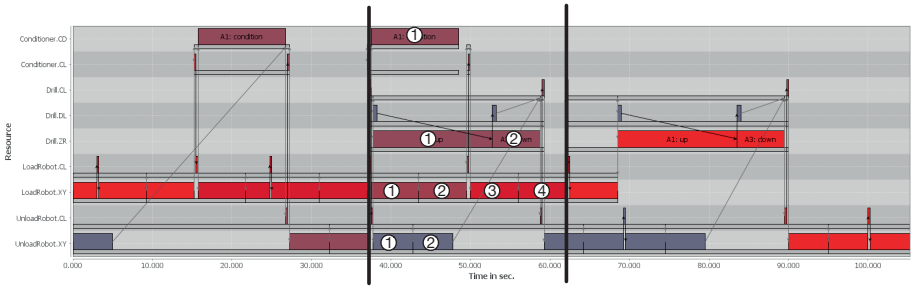


Figure 5.5: *Result of stochastic criticality analysis.*

Table 5.5 shows the resulting node criticality values of the classical critical path approaches, CPM for best (B), worst (W), most likely (L) cases and PERT. In the case of CPM, the critical path and critical nodes are computed for the best, worst and most likely executions times. In case of PERT, we first compute the "expected" execution times using equation $(B + 4 \times L + W)/6$ as in [55] with the values in Table 5.4 and then find the critical path and critical nodes assuming "expected" execution times. Table 5.6 depicts the Criticality Indices and the associated absolute error bound of the estimation obtained via our SCA approach. The running time of the SCA approach for this example (using a confidence level of 99% and an 0.01 error bound), is below a minute and requires 18700 iterations to converge. Observing the two tables, it can be concluded that the PERT approach yields a less informative overview of the criticality of the actions, compared to the SCA results. According to the PERT results, the Conditioner has a criticality of 0, and the only actions in the critical path are LR actions. Consequently, to reduce the overall makespan one would improve the execution times of the LR actions. However, the SCA results show that it is also relevant to consider the Conditioner action since it has a CI of 0.3221. Therefore, further improvements to the makespan of the system can be achieved by improving the execution of the Conditioner actions. Furthermore, even if we consider all the information given by the CPM cases together (i.e. the best, worst, and most-likely cases), we do not obtain as much information as we with do the SCA approach. Observe that in none of the three cases the actions of Conditioner occur in the critical path, while the SCA indicates a CI of 0.3221.

Table 5.6: *Criticality Indices obtained from SCA approach for the aggregated actions of a peripheral.*

Actions p/ peripheral	CI	Abs. Error
COND.H	0.3221	0.0009
DRILL.D (1,2)	0.3312	0.0009
LR.R and LR.Z (1,2)	0.3467	0.0008
LR.R and LR.Z (3,4)	0.6821	0.0008
UR.R and LR.Z (1,2)	0.0000	0.0008

Using the classic methods CPM and PERT, we would also have identified the Conditioner to be a bottleneck in a next design cycle, after having optimized the critical peripherals actions. By using our SCA approach we obtain more information about all potential improvements at once and these can be ranked according to their Criticality Index. In the example the SCA discovered a candidate to improve that none of the traditional methods found, although our method did not assign it the highest rank. Situations could even occur in which the candidate ranked highest in SCA is discovered by none of the traditional methods.

5.6 Related Work

The study of criticality of a path, or a task in a path, has it origins in the Critical Path Method (CPM) [47] and in the Project Evaluation and Review Technique (PERT) [54], [55]. Projects are modeled as activity networks, where nodes represent tasks, annotated with (deterministic (CPM) or stochastic (PERT)) durations. Edges model dependencies between tasks and a critical path determines the total completion time of the project. The use of critical path based methods has made its way into different fields such as embedded systems and IC design, for deterministic execution times [11], [20], [42] as well as for stochastic execution times [77], [88]. Although both CPM and PERT focus on the criticality of paths, is has been shown that is it more meaningful to rely on the Criticality Index (CI) metric [21], [29], [32], [83]. Intuitively, the goal of CPM or PERT is to give feedback on which tasks to focus to minimize the lead time of a project. The CI metric allows the ranking of the tasks within a project given the likelihood that they are on a critical path. We follow the Criticality Index approach in this chapter. Determination of the CI can be done either: 1) analytically [29], [49], [56] or 2) by Monte-Carlo estimation approaches [21], [75], [83]. For the general case, analytical solutions are computationally demanding and cannot handle large-sized stochastic activity networks [86]. To obtain scalable analysis our approach relies on simulation. In this chapter we take inspiration from the field of project planning and bring the notion of Criticality Index (CI) to the domain of flexible manufacturing system such that it can be used to improve the information obtained by bottleneck analyses. This allows identification of performance bottlenecks in a given activity sequence by ranking the CIs of all nodes. Our main contribution is the formalization of the Stochastic Criticality Analysis (SCA) approach with formal mathematical support together with confidence intervals to obtain results with known accuracy.

5.7 Conclusions

In this chapter we discussed the identification of performance bottlenecks during the design exploration of flexible manufacturing systems. This corresponds to the last step in the design exploration and optimization framework proposed in this thesis. To capture the stochastic nature of physical actions of these systems (i.e. actions that are susceptible to variations in their execution times) we extended our specification framework with stochastically timed actions and activities. Further, we introduced Stochastic Criticality Analysis (SCA) as a means to identify bottlenecks in a given activity sequence. This analysis technique estimates the Criticality Index (CI) of every node of a sequenced activity. The CI of a node corresponds to the likelihood of that node being on the critical path. This estimation is achieved via a Monte-Carlo simulation approach which we formalized in this chapter. Furthermore, we extended the SCA approach by using confidence intervals to obtain estimation results with known accuracy.We demonstrated the SCA analysis by identifying performance bottlenecks and possible design improvements in a variant of the Twilight system for which actions exhibit stochastic execution times. We concluded that the SCA analysis provides more informative results in a single step than the traditional critical path analysis commonly used for bottleneck design such as the Critical Path Method (CPM) or the Project Review and Evaluation Technique (PERT). As a consequence, fewer design iterations are expected to be require to converge to a final design that either satisfies performance requirements or for which no further improvements are possible.

6 | Case Study: Wafer Handling

This chapter presents a case study in which we apply our framework to specify a wafer handling controller, optimize its makespan and identify performance bottlenecks. The wafer handling controller is one of the controllers of an ASML TWINSCAN™ manufacturing system which executes the photo-lithography step in the semiconductor manufacturing process. This controller is responsible for the management of wafer logistics within the whole system. Its purpose is to ensure that each silicon wafer of a batch of wafers is correctly conditioned and aligned for the photo-lithography process (which exposes a circuit pattern onto the silicon wafer) and that the completion time of a batch of wafers is minimized. We start by a specification of the plant, activities and logistic requirements of the system using the concepts introduced in Chapter 2. Next we find the optimal makespan activity sequence for a batch of wafers using the technique of Chapter 3. Even though for this large case study the optimal makespan activity sequence can be computed without heuristics, we still show the effectiveness of the algebra of logistics automata of Chapter 4 by defining an additional requirement to significantly reduce the optimization-space (in this case even without losing optimality). Finally we select candidates for design improvement by identifying performance bottlenecks in the makespan optimal activity sequence using the Stochastic Criticality Analysis technique of Chapter 5. For confidentiality reasons, we omit the detailed listing of system actions and their execution times and do not provide a complete specification of the resources and peripherals of the system.

This chapter is organized as follows. Section 6.1 provides an overview of the lithography process. Section 6.2 introduces the wafer handling controller, its plant specification, activities and logistics requirements. Section 6.3 discusses the optimization of the wafer handling controller for a batch of 25 products and Section 6.4 discusses possible design improvements by identifying performance

bottlenecks of the makespan optimal activity sequence. Finally Section 6.5 concludes the chapter.

6.1 Photo-lithography

In recent years the number of computing devices in our daily lives and their capabilities have grown extremely fast. This growth closely follows the trend popularized by Moore's law [60] stating that the transistor density in an Integrated Circuit (IC) doubles roughly every 18-24 months. The reduction in transistor size is mainly due to the quality and accuracy at which the structures that define a transistor are transfered to a silicon substrate. This is usually done by a photo-lithography process step in semiconductor manufacturing, which is realized by systems such as the lithography scanners produced by ASML. In essence, lithography is the process of transferring a certain pattern from one surface to another. When the transferring method is based on light, this process is called photo-lithography. Semiconductor manufacturing systems use photo-lithography to image circuit patterns at a nanometer scale onto silicon wafers. Light from a light source passes through a patterned quartz plate called a reticle. The reticle contains a scaled image of the desired circuit pattern. The projection passes through a reduction lens and is then projected onto the wafer. The wafer surface is coated with a photosensitive material that hardens when exposed to ultra-violet light. As a result, after exposure the circuit design is imprinted on the wafer as a pattern of hardened photosensitive material. The exposed wafer is then developed, etched (i.e. removing unprotected material by polishing) and post-processed with several other chemical processes.

Modern circuits require multiple of these exposed wafer layers to form complex circuit connections and structures, and therefore the described process must be repeated several times. In many cases it might require as many as 40 of these layers. At the end, the final wafer is split into individual chips that after packaging are deployed into electronic devices. There are two main types of semiconductor manufacturing systems performing this process, steppers and scanners. A stepper exposes the complete reticle image while a scanner only exposes a narrow stripe that is as wide as the reticle but only a fraction of its length [23]. Exposing of the circuit pattern by scanning is achieved by moving the reticle and wafer in opposite directions in a synchronized way. The movements executed during scanning easily reach accelerations of 20G while delivering nanometer accuracy on the exposure [25]. To improve accuracy a

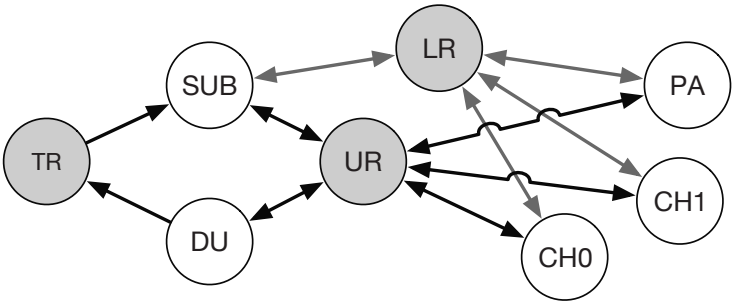


Figure 6.1: *Routing and resources of the Wafer Logistics.*

measure operation usually precedes the expose operation. During measure, the surface of the wafer is analyzed to detect irregularities and deformations. This information is then used to improve the quality and accuracy of the expose operation.

The ASML TWINSCAN™ lithography scanner is equipped with dual-stage scanning. In this type of scanner 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 operation or the expose operation. In this chapter we focus on a controller of a scanner-based semiconductor manufacturing system, the ASML TWINSCAN™ wafer handling controller.

6.2 The Wafer Handling Controller

A lithography manufacturing system such as an ASML TWINSCAN™ is a large system composed of several controllers responsible for different aspects of the manufacturing process. In this chapter we will focus on the the wafer handling controller. This controller manages the logistics of wafers through the system by assigning operations to resources and determining their order. Its goal is to ensure that each wafer in a batch follows its life-cycle and that the completion time of the batch is minimized.

The wafer handling controller is responsible for two pre-processing operations: conditioning and alignment. The wafer surface should be within a specific range of temperatures. Any deviation from this range might degrade the quality of the expose operation. Therefore the wafer must be conditioned to a correct

temperature before the exposure operation starts. Further, multiple exposure operations are required on the same wafer to build a complete integrated circuit. To minimize the (overlay) error between two consecutive exposed circuit patterns each wafer must be accurately aligned. Besides these pre-processing operations, the wafer handling controller is also responsible for the two chucks of the dual-stage scanner. The scanner has two locations for the chucks, a measure location and an expose location. Each chuck must move to the measure location for the measure operation and to the expose location for the exposure operation. Moreover, the unloading and loading of a wafer from and to a chuck is only performed at the measure location. The scanner utilizes an immersion technology to improve the expose operation. This immersion technology enforces that a chuck can never be without a wafer with the risk of disrupting the process. To avoid this, the scanner is equipped with several *closing* wafers. These wafers do not require any pre-processing or exposure and are always loaded in the scanner when the system is not in production.

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). A FOUP is a specialized plastic enclosure designed to hold silicon wafers securely and to allow their transfer between different machines for different phases of the chip manufacturing process [30]. In the next section we describe the system in terms of its resources, peripherals, activities, logistics and system requirements. We omit any specification in terms of locations, layout, actions and execution times for confidentially reasons.

6.2.1 Resources and Peripherals

For wafer handling eight resources are utilized which are depicted in Figure 6.1 by annotated circles. Circles with a white background represent the production resources SUB, DU, PA, CH0 and CH1, while darkened circles represent transport resources 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 predefine temperature. The Pre-Aligner (PA) accurately aligns a wafer with respect to a reference position. The Discharge Unit (DU) is an output buffer. The Track (TR) is an external 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 6.1 lists all the eight resources utilized by wafer handling and their corresponding peripherals. Both the LR and UR robots are composed of two peripherals; a clamp (LR.CL and UR.CL) to clamp and unclamp a wafer and a motor peripheral (LR.MOTOR and UR.MOTOR). The motor peripheral is able to perform three-dimensional movements. This enables the transport of wafers across different resources using the LR and UR resources. Figure 6.1 depicts the possible movements of wafers between resources by the TR, LR and UR resources using directional arrows. For instance, the LR resource is able to pick and place a wafer from the SUB, PA, CH0 and CH1 resources. The SUB is composed of a clamp (SUB.CL) and a conditioner (SUB.COND) peripheral to correct deviations in the wafer temperature. The PA is composed of a clamp (PA.CL) and an aligner (PA.ALIGNER) peripheral to adjust the orientation of a wafer. Both chuck resources CH0 and CH1 are composed of a clamp peripheral (CH0.CL and CH1.CL) and a wafer stage (CH0.WS and CH1.WS) peripheral. The wafer stage enables the measure and expose operations and moves the chucks between the measure location and the expose location. The TR is composed of a clamp (TR.CL) and a robot (TR.ROBOT) peripheral to input and output wafers from the system. Finally, the DU has a clamp peripheral (DU.CL) to clamp and unclamp a wafer before it is outputted to the track resource.

6.2.2 Activities

Table 6.2 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

Table 6.1: *Wafer handler resources and peripherals.*

Resource	Peripherals
(LR) Load Robot	LR.MOTOR and LR.CL
(LR) Unload Robot	UR.MOTOR and UR.CL
(DU) Discharge Unit	DU.CL
(SUB) Storage Unit	SUB.COND and SUB.CL
(PA) Pre-Alginer	PA.ALIGNER and PA.CL
(CH0) Chuck 0	CH0.WS and CH0.CL
(CH1) Chuck 1	CH1.WS and CH1.CL
(TR) Track	TR.ROBOT and TR.CL

6.1, 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 6.2 (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.

Table 6.2: *Set of activities for our case study.*

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

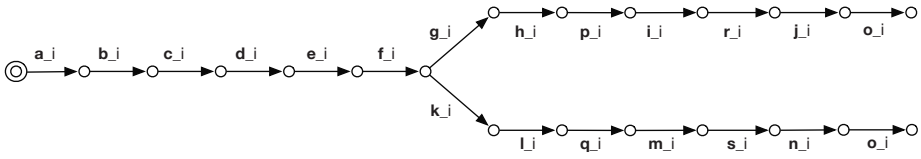


Figure 6.2: *Logistics automaton L_{LC_i} capturing the life-cycle for a wafer i of a batch of wafers.*

6.2.3 Wafer Logistics

For every wafer we define its life-cycle (wafer product flow). Figure 6.2 depicts logistics automata L_{LC_i} which capture the life-cycle of wafer i of a batch of wafers. The production life-cycle (captured by L_{LC_i}) 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 (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 resource the product flow remains similar, however, in that case activities for the CH1 resource would be executed **k**_i, **l**_i, **q**_i, **m**_i, **s**_i and **n**_i.

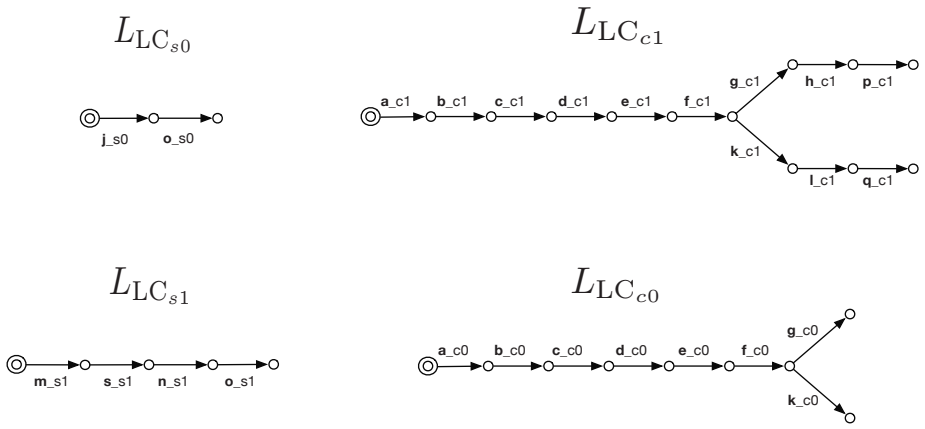


Figure 6.3: *Logistics automata $L_{LC_{s0}}$, $L_{LC_{s1}}$, $L_{LC_{c1}}$ and $L_{LC_{c0}}$ capturing the life-cycle for the closing wafers s0, s1, c1 and c0 respectively.*

Closing wafers and assumptions

In this section we discuss the assumptions made with respect to the initial and final state of the system, and provide the final specification of the logistics requirements for a FOUP.

The scanner utilizes an immersion technology that enforces that chuck resources CH0 and CH1 can never be empty (i.e. there must always be a wafer on a chuck). Whenever the system is not in production, closing wafers are loaded on the chuck resources. For this case-study we take this into account by modeling four closing wafers s0, s1, c0 and c1. Closing wafers s0 and s1 represent the closing wafers already in the system before the processing of a FOUP, and closing wafers c0 and c1 represent the closing wafers that should be loaded once the FOUP is processed. For simplicity, we assume that these wafers can be exposed and measured as if they were production wafers, even though this is not necessary.

Since the scanner has two chucks that can be in the same positions, the system could start with CH0 at the measure location and CH1 at the expose location, or vice-versa. In this case study, 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 closing wafers s0 and s1 captured by logistics automata $L_{\mathrm{LC}_{s0}}$ and $L_{\mathrm{LC}_{s1}}$ depicted in Figure 6.3. The life-cycle of s0 (captured by $L_{\mathrm{LC}_{s0}}$) starts

with the unloading of the closing 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 (captured by $L_{\mathrm{LC}_{s1}}$) starts with the expose activity of closing 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 closing wafers c0 and c1 follows the same flow as the life-cycle of a production wafer (captured by automata L_{LC_i}) but ends in different final states. These are captured by logistics automata $L_{\mathrm{LC}_{c0}}$ and $L_{\mathrm{LC}_{c1}}$ depicted in Figure 6.3. The life-cycle of closing wafer c0 (captured by $L_{\mathrm{LC}_{c0}}$) ends once the closing wafer is loaded onto either the CH0 resource or the CH1 resource (activity **g**_c0 or **k**_c0). The life-cycle of closing wafer c1 (captured by $L_{\mathrm{LC}_{c1}}$) ends once the closing wafer is moved to the expose location placed either on the CH0 resource and CH1 resource (activity **p**_c1 or **q**_c1). The specified life-cycles for closing wafers c0 and c1 ensure that there is a wafer on each chuck resource once the system ends production.

Even though we enforce a specific ordering by specifying the initial position of the chucks, we decide to maintain the automata capturing their life-cycle generic for clarity and re-utilization purposes (with the exception of automata $L_{\mathrm{LC}_{s0}}$ and $L_{\mathrm{LC}_{s1}}$). If desired, the initial state of the scanner (the initial position of resources CH0 and CH1) can be altered by modifying the life cycle of automata $L_{\mathrm{LC}_{s0}}$ and $L_{\mathrm{LC}_{s1}}$.

Now that we have introduce the life-cycle of a production wafer in a FOUP (automata L_{LC_i}) and of the closing wafers (automata $L_{\mathrm{LC}_{s0}}$, $L_{\mathrm{LC}_{s1}}$, $L_{\mathrm{LC}_{c0}}$ and $L_{\mathrm{LC}_{c1}}$) we are ready to specify the logistics of a FOUP. The batch specification of a FOUP 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 the logistics automaton for a FOUP as L_{LC}. Note that L_{LC} is np-repulsing and that by Lemma 4.20 the composition of all product automata is then also np-repulsing.

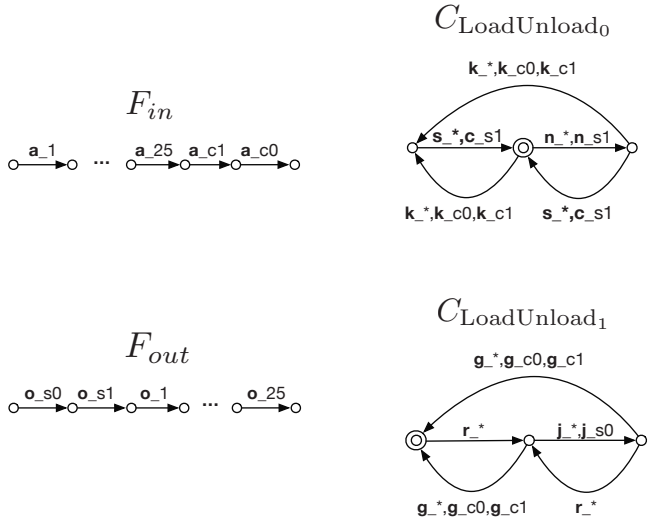


Figure 6.4: *Constraint automata F_{in} and F_{out} capturing the FIFO input and output ordering and constraint automata $C_{UnloadLoad_0}$ and $C_{UnloadLoad_1}$ enforcing that the load and unload of wafers occurs at the measure location for CH0 and CH1 respectively.*

6.2.4 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:

- *Products shall enter and leave the system in a First-In-First-Out (FIFO) order.*

A FOUP contains around 25 wafers that need to be exposed to a particular circuit design. For most integrated circuits this process needs to be repeated multiple times. For quality reasons the physical conditions under which each expose operation occurs should be kept the same. In order to maintain quality and track possible errors for each wafer in a FOUP it is important that the order in which they are stored is kept the same. For this reason a FIFO ordering is imposed on the wafer handling controller. This is captured by constraint automata F_{in} and F_{out} depicted in Figure 6.4. Constraint F_{in} enforces an order on the input of wafers into the scanner. This is done by ordering activities **a** for all wafers in the FOUP and closing wafers c0 and c1. Constraint F_{out} enforces an order on the output of wafers from the scanner. This is done by

ordering activities **o** for all wafers in the FOUP and closing wafers s0 and s1. Together they impose a FIFO ordering of the wafers in a FOUP. Note that these constraint automata are p-attracting.

- *There shall only be one product at a time in each resource (unary capacity).*
- *Wafers shall not collide (i.e. products shall not be be placed on an occupied resource).*

All the resources used by the wafer handling controller can only hold one wafer at a time. Any violation of this constraint results in wafer damage. To ensure that this does not happen we enforce 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 constraint automata $C_{\text{SUB}}, C_{\text{UR}}, C_{\text{LR}}, C_{\text{PA}}, C_{\text{CH0}}$ and C_{CH1} depicted in Figure 6.5. For example, consider automata C_{SUB}. The initial state represents the resource to be empty and the other state that 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. This way it is ensured that the requirement is never violated. Note that all these constraint automata are p-attracting.

- *Chucks must swap positions in between every measure and expose activity (Swap)*

Recall that we are specifying the requirements of a dual-stage scanner where two chuck resources, CH0 and CH1, are able to perform both the measure and expose activities. After these activities are executed, the chuck resource on the measure location must move to the expose location, and the chuck resource on the expose location must move to the measure location. Further, in other to unload and load a wafer from and to the CH0 and CH1 resources the resources must be at the measure location. These requirements are partially enforced by the logistics product flow L_{LC_i}. However, this does not ensure that the same holds across different wafers in the batch. For instance, without this requirement, it is possible that a wafer loaded on the CH0 resource

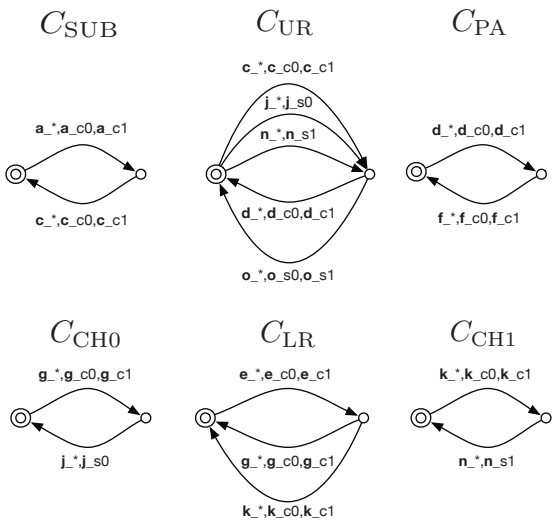


Figure 6.5: *Constraint automata capturing capacity requirements for the wafer handler resources.*

and another wafer loaded on the CH1 resource perform the expose activity simultaneously. However, this would imply that CH0 and CH1 resources are both at the expose location, which should not happen. These requirements are captured in our specification by constraint automata C_{Swap} depicted in Figure 6.6, and $C_{\mathrm{UnloadLoad}_0}$ and $C_{\mathrm{UnloadLoad}_1}$ depicted in Figure 6.4. Note that these constraint is p-attracting.

Constraint automaton C_{Swap} enforces that after every execution of a measure and expose activity chuck resources CH0 and CH1 must 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 where an expose activity is performed by the CH1 resource (activities **m**_* and **m**_s1) and a measure activity is performed by the CH0 resource (activities **h**_* or **h**_c1). Situation 2) is captured by top branches where where an expose activity is performed by the CH0 resource (activities **i**_*) and a measure activity is performed by the CH0 resource (activities **l**_* or **l**_c1). 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

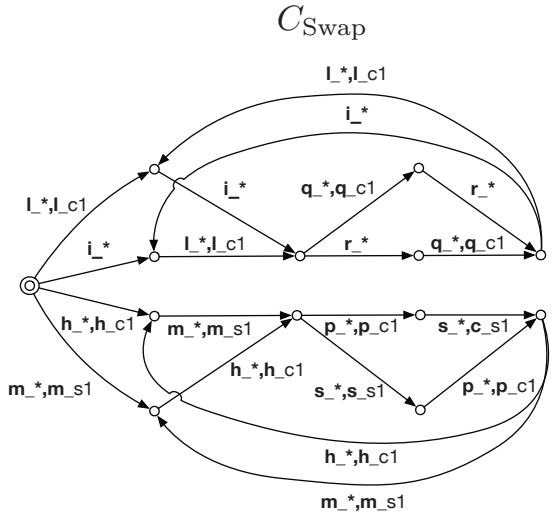


Figure 6.6: *Constraint automata C_{Swap} capturing the exchange of chuck positions after the measure and expose operations.*

(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).

Constraint automata $C_{\text{UnloadLoad}_0}$ and $C_{\text{UnloadLoad}_1}$ 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 $C_{\text{UnloadLoad}_0}$), 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 (captured by $C_{\text{UnloadLoad}_1}$) with the difference that in this case the initial state only enables the movement to the measure location, since the CH1 resource starts at the expose location (following the assumptions made in Section 6.2.3).

Table 6.3: *Size of the state-space and optimization-space, and computed makespan for a batch of* 25 *wafers and* 4 *closing 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

- *Wafers shall be aligned correctly and conditioned to a desired temperature before being placed on a chuck (Overlay).*

This requirement is guaranteed by the life-cycle automata L_{LC_i} which ensures that the conditioning and alignment of a wafer (activities **b**_i and **e**_i) are always done before being loaded on a chuck (activities **g**_i and **k**_i).

The specification for a FOUP processed by the wafer handling controller can then be obtained by constraining logistics automaton L_{LC} with all these above described system constraint. This would result in automaton: $L_{\text{LC}} \upharpoonright C_{\text{COND}} \upharpoonright C_{\text{UR}} \upharpoonright C_{\text{LR}} \upharpoonright C_{\text{PA}} \upharpoonright C_{\text{CH0}} \upharpoonright C_{\text{CH1}} \upharpoonright F_{in} \upharpoonright F_{out} \upharpoonright C_{\text{Swap}} \upharpoonright C_{\text{UnloadLoad}_1} \upharpoonright C_{\text{UnloadLoad}_1}$. In the next section we discuss the makespan optimization of a FOUP by using the specification style and heuristic approach of Chapter 4.

6.3 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 FOUP of 25 wafers (including the 4 closing wafers $s0$, $s1$, $c0$ and $c1$). The experimental setup is the same as used to optimize the Buffered Twilight system in Section 4.7. Table 6.3 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. Recall that we let L_{LC} denote the logistics automaton for a FOUP defined as

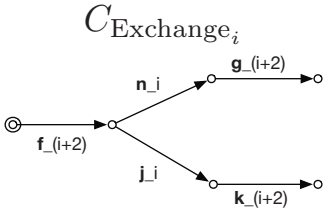


Figure 6.7: *Constraint automata $C_{Exchange}$ capturing a specific ordering for the exchange of wafers from/to the chuck resources.*

$L_{\text{LC}_{s0}} \odot L_{\text{LC}_{s1}} \odot (L_{\text{LC}_1} \odot L_{\text{LC}_2} \odot \cdots \odot L_{\text{LC}_{25}}) \odot L_{\text{LC}_{c1} \odot L_{\text{LC}_{c0}}}$ is np-repulsing.

The first row (Logistics) of Table 6.3 represents L_{LC}. Each additional row of the table represents the cumulative addition of another requirement (indicated by the $+$ symbol). Where +Capacity refers to automaton $(L_{\text{LC}} \restriction C_{\text{COND}} \restriction C_{\text{UR}} \restriction C_{\text{LR}} \restriction C_{\text{PA}} \restriction C_{\text{CH0}} \restriction C_{\text{CH1}}$, +FIFO to automaton $(L_{\text{LC}} \restriction C_{\text{COND}} \restriction C_{\text{UR}} \restriction C_{\text{LR}} \restriction C_{\text{PA}} \restriction C_{\text{CH0}} \restriction C_{\text{CH1}}) \restriction F_{in} \restriction F_{out}$ and +Swap to automaton $(L_{\text{LC}} \restriction C_{\text{COND}} \restriction C_{\text{UR}} \restriction C_{\text{LR}} \restriction C_{\text{PA}} \restriction C_{\text{CH0}} \restriction C_{\text{CH1}} \restriction F_{in} \restriction F_{out}) \restriction C_{\text{Swap}} \restriction C_{\text{UnloadLoad}_1} \restriction C_{\text{UnloadLoad}_1}$. 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 yielding a minimal makespan of 331.7 seconds can be computed. For the specifications for which we are able to compute the logistics automaton, a solution to the BMO problem is found in under 2 minutes.

Even though it is sufficient to apply all system constraints to compute the makespan optimal activity sequence, we will demonstrate the effectiveness of the heuristics approach of Chapter 4 by further pruning the state-space and optimization-space. Recall that by Lemma 4.21 we know that the constraining of L_{LC} with constraints C_{SUB}, C_{UR}, C_{LR}, C_{PA}, C_{CH0}, C_{CH1}, F_{in}, F_{out}, C_{Swap}, $C_{\text{UnloadLoad}_1}$ and $C_{\text{UnloadLoad}_1}$ yields a np-repulsing automaton. This is an important result since by Theorem 4.23 we know that the optimization-space is pruned by further constraining of L_{LC} with a p-attracting constraint automata. To illustrate this we will use over-specification by subsequently formalizing a non-essential requirement as constraint automaton C_{Exchange} depicted in Figure 6.7.

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 $i+2$. There are many possible activity orderings to exchange wafers from and to chucks CH0 and CH1, but it is known from domain-knowledge that it is usually done in the following way. First the currently aligned wafer $i+2$ at the PA resource is picked by the LR (activity **f**_i). Then immediately after wafer i is picked from either CH0 or CH1 by the UR resource (activities **j**_i or **n**_i) the aligned wafer on the LR is placed on chuck resource CH0 or CH1 (activities **g**_i or **k**_i) which is now empty. Due to this, we figured that a proper heuristic is defined by constraint automaton C_{Exchange_i} (depicted in Figure 6.4) 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 then instantiated for each wafer i where $1 \leq i \leq 25$. For the last two closing wafers this requirement is not enforced. We apply these constraints to obtain: $(L_{\text{LC}} \upharpoonright C_{\text{SUB}} \upharpoonright C_{\text{UR}} \upharpoonright C_{\text{LR}} \upharpoonright C_{\text{PA}} \upharpoonright C_{\text{CH0}} \upharpoonright C_{\text{CH1}} \upharpoonright F_{in} \upharpoonright F_{out} \upharpoonright C_{\text{Swap}} \upharpoonright C_{\text{UnloadLoad}_1} \upharpoonright C_{\text{UnloadLoad}_1}) \upharpoonright C_{\text{Exchange}_1} \upharpoonright \cdots \upharpoonright C_{\text{Exchange}_{23}}$. Notice C_{Exchange} is a p-attracting automata. Since $L_{\text{LC}} \upharpoonright C_{\text{SUB}} \upharpoonright C_{\text{UR}} \upharpoonright C_{\text{LR}} \upharpoonright C_{\text{PA}} \upharpoonright C_{\text{CH0}} \upharpoonright C_{\text{CH1}} \upharpoonright F_{in} \upharpoonright F_{out} \upharpoonright C_{\text{Swap}} \upharpoonright C_{\text{UnloadLoad}_1} \upharpoonright C_{\text{UnloadLoad}_1}$ is np-repulsing we know by Theorem 4.23 that the constraining will reduce the optimization-space. This is confirmed by the results shown in Table 6.3 in the +Exchange row. Notice that in this case the optimal solution could already be found without over-specification. Because of this 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.

On the design of requirement heuristics

Determining the additional requirements that constitute an effective heuristic to prune the optimization-space is not straight-forward. However, we can provide a few insights on how these can be found. It is often the case that these are very system specific and therefore it is useful to always consider

Table 6.4: *PERT distributions for the stochastic action of the wafer handler*

Peripheral	min	max	mode	λ
PA.A.align	0.9	1.5	1	6
SUB.H.condition	5.8	6.5	6	6
CH0/1.WS.measure	7.6	8.1	7.9	40
CH0/1.WS.expose	7.9	8.1	8	40

domain-knowledge or similar systems to determine heuristics as over-specified requirements. This is the approach followed in the Buffered Twilight of Chapter 4 and also how we came to constraint C_{Exchange} for the wafer handling controller. Furthermore, it is also often the case that "safe" constraints (constraints that will preserve optimality) can be found by looking at combinations of activities that do not share any resources. The order in which these activities are executed has no impact on the makespan [72]. This is visible in the case of constraint C_{Exchange}, where we enforce a specific ordering between activities **f** and **n**, and **f** and **j**.

6.4 Identifying Performance Bottlenecks

In the previous section we solved the BMO for the logistics automaton of the wafer handling controller for a typical FOUP of 25 wafers and 4 closing wafers. We found a makespan optimal activity sequence for which the makespan of a FOUP equals to 331.7. In this section we apply the SCA approach, introduced in Chapter 5, to identify performance bottlenecks in the optimal activity sequence. For this purpose we sequence the optimal activity sequence using the (;) sequencing operator and then apply the SCA approach assuming a confidence level of 99% and an error bound of 0.01. Activities Condition (**b**), Pre-Align (**e**), Measure (**h,l**) and Expose (**i,m**) include actions which capture physical processes exhibiting variations on their execution times.

We assume a PERT distribution for each of these actions for which the parameters are described in Table 6.4. The results for this configuration were obtained using an experimental setup consisting of an Intel i5-4590 CPU @3.30 GHz and with 8.00 GB of memory. For this case study, the SCA approach converges in 10760 runs taking around 15 minutes.

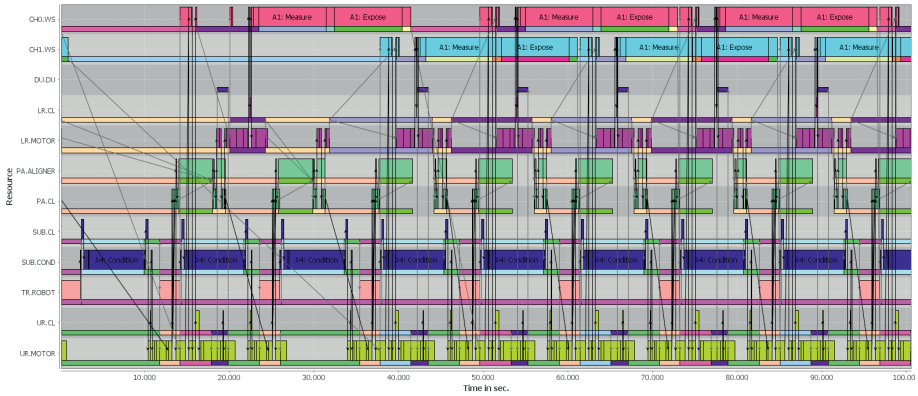


Figure 6.8: *Gantt chart of the optimal activity sequence for a batch of 25 wafers and 4 closing wafers.*

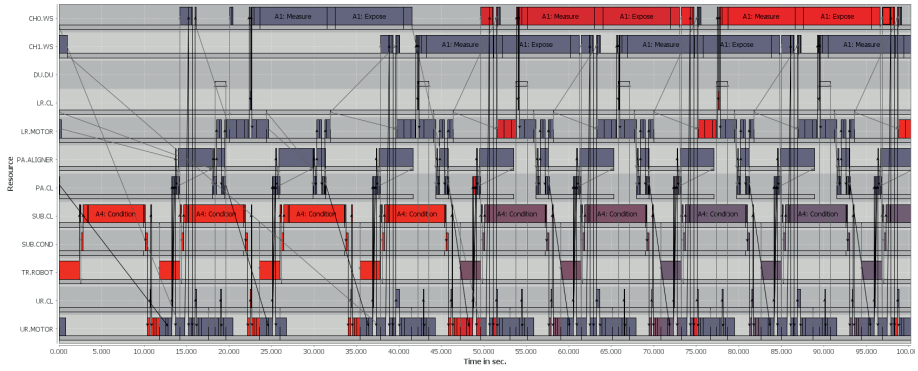


Figure 6.9: *Gantt chart resulting from the SCA approach for the optimal activity sequence for a batch of 25 wafers and 4 closing wafers.*

Figure 6.8 depicts the Gantt chart of the sequenced makespan-optimal activity sequence and Figure 6.9 the result of the SCA analysis. The horizontal axis represents the actions executed in time and the vertical axis shows the horizontal lanes for each peripheral. Each horizontal lane of the Gantt chart depicts two types of boxes. Thick boxes represent the execution of an action by a peripheral and slim boxes represent the time that the corresponding resource is claimed by an activity. In Figure 6.9, different shades of red coloring within thick boxes reflect different criticality indices. The lighter the redness the lower the criticality index, and vice-versa. For understandability, Figure 6.9 depicts only a snippet of the full execution of the activity, since the full Gantt chart would not be easily readable. This snippet captures the processing of a FOUP from 0 to 100 seconds of system execution. Further, we do not list the full ranking per action but rather summarize the critical peripheral actions in the context of a peripheral and the activity to which these belong to.

Observing the figure we conclude that during start-up (from 0 to approximately 55 seconds) actions of the TR.ROBOT peripheral (for activity Track_2_SUB) and of the SUB.CL and SUB.COND peripherals (for activity SUB_Condition) exhibit high criticality since they are colored with a bright shade of red. The actual criticality index of these actions is 1.00. This implies that independent of the timing of the peripheral actions, the actions of the TR.ROBOT and SUB.CL peripherals are always on the critical path.

For the remainder of the captured system execution (steady-state, 55 seconds onwards), the criticality of different actions is not so clear as during start-up. In this case, we see the following ranking:

- actions of peripherals SUB.COND and SUB.CL (for activity SUB_Condition) and TR.ROBOT (for activity Track_2_SUB) have an estimated criticality index which varies between 0.11 and 0.21.
- actions of peripheral UR.MOTOR (for activity SUB_2_UR) have an estimated criticality index which varies between 0.32 and 0.74.
- actions of peripherals LR.MOTOR (for activity LR_PA_CH0) and CH0.WS (for activities CH0_Measure and CH0_Expose) have an estimated criticality index which varies between 0.74 and 0.98.

This overview allows to rank and evaluate possible improvements on the design specification. For the start-up of the system, it is clear that improvements can be achieved by speeding the TR.ROBOT movements and reducing the execution time of the condition operation of the SUB.COND peripheral.

With respect to the steady-state execution, the most effective improvement would be achieved by improving the execution of the CH0.WS and LR.MOTOR peripheral actions. The CH0.WS peripheral is responsible for the measure and expose operations and the LR.MOTOR, in this context, is responsible for the loading of a wafer onto CH0 or CH1 resources. However, for the optimization of wafer logistics the execution times of the measure and expose actions are usually given and not under the control of the wafer handling controller. Therefore, it is acceptable for the CH0.WS and CH1.WS peripherals to be a performance bottleneck. Under these circumstance a good improvement would be to speed up peripheral LR.MOTOR. As an experiment to validate this hypothesis, we double the speed of the movements of the LR.MOTOR peripheral in a new specification of the wafer handling controller and run the optimization approach on the modified specification. Doing so yields a minimal makespan of 316.7 (an improvement of 5% over the initial specification). Figure 6.10 depicts Gantt chart of the execution of the sequenced activity sequence for the modified specification and Figure 6.11 the SCA results. We can observe that by speeding up the LR.MOTOR peripheral actions these actions are no longer performance bottlenecks. In the modified specification, the performance bottlenecks are now only the peripheral actions of SUB.COND and SUB.CL (for activity SUB_Condition), TR.ROBOT (for activity Track_2_SUB) and UR.MOTOR (for activity SUB_2_UR) which now exhibit a criticality index of 1.00.

6.5 Conclusions

In this chapter we applied all the analyses and techniques of the framework introduced in this thesis to specify and optimize an industrial wafer handling controller. Using the concepts of Chapter 2, we demonstrated how to decompose an industrial system into a plant of resources and peripherals and a set of activities capturing the operations required for the manufacturing of a batch of wafers. Moreover we captured the wafer flow and system requirements of the wafer handler as a set of logistics and constrain automata. We showed how to compute a solution to the BMO of the specified logistics automaton of the wafer handling controller by using the modular approach introduced in Chapter 4. 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.

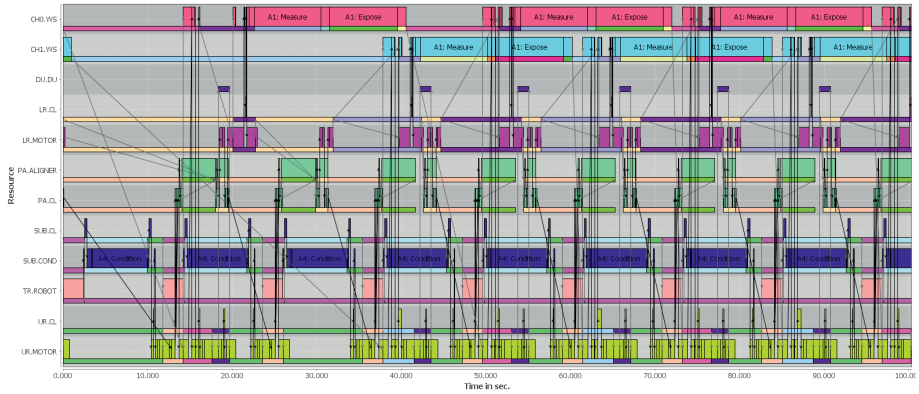


Figure 6.10: *Gantt chart of the modified specification where we double the speed of the LR.MOTOR peripheral actions.*

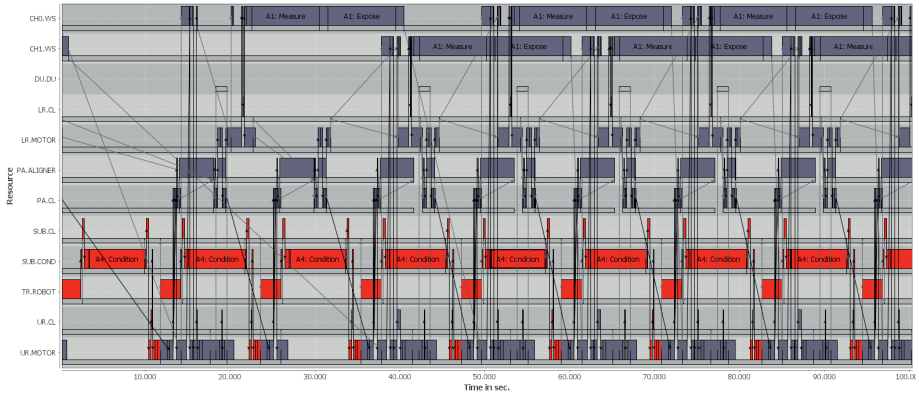


Figure 6.11: *Gantt chart resulting from the SCA approach for the modified specification where we double the speed of the LR.MOTOR peripheral actions.*

This reduction demonstrated the effectiveness of using over-specification as a means to prune the optimization-space of the logistics of flexible manufacturing systems.

We explored possible improvements to the specification of the wafer handler by identifying performance bottlenecks in the computed makespan optimal activity sequence. From the identified bottlenecks we discussed possible improvement candidates that could improve the makespan. This chapter illustrated by means of a case study that our design framework can be used as a systematic method for the design exploration and optimization of flexible manufacturing systems. Similar studies are being done at ASML to perform layout analysis of a given specification by modifying the placement or number of resources used. These studies are executed using a proprietary tool developed by TNO-ESI and ASML which integrated several results of this thesis.

7 | Conclusions and Future Work

Globalization and current trends in the consumer goods market have changed the landscape of the manufacturing industry. This industry faces a very competitive market where manufacturing systems need to adapt faster to changes in demand, shorter time-to-market and allow for product customization. As a solution to cope with these trends, manufacturers turned to flexible manufacturing systems. Flexibility is achieved by adding system functionality, for example, in the form of multiple product routes within the system or by developing production units able to perform different operations on products. By becoming more flexible and allowing custom and multiple product types to be manufactured within the same system, manufacturing systems are able to meet these market demands with a minimum impact on cost. However, this leads to more involved system specifications, which need to consider that operations can be executed using multiple production units (resource sharing) and different product routes. This added flexibility also makes the productivity of these systems highly dependent on aspects such as the geometrical layout, the routes of products between different resources and the mix of product types manufactured. Depending on the distances of resources and viability of routes between them, traveling times of products change. Different product types being manufactured on the same system could imply the need to account for setup-times (i.e. amount of time required for a resource to adapt and perform a different operation). These aspects make the prediction of the productivity of a specification and the evaluation of different design choices more complicated. This thesis proposed several contributions addressing the challenges in the specification and design exploration of flexible manufacturing systems. In this chapter we present a summary of the conclusions of this thesis and discuss future work.

The remainder of this chapter is organized as follows. Section 7.1 discusses the main contributions and conclusions of this thesis and Section 7.2 discusses future work.

7.1 Conclusions

This thesis set out to develop a systematic approach for the specification and design exploration of complex Flexible Manufacturing Systems (FMSs). This approach should address several of the challenges in the specification and design exploration of these systems, as identified in Section 1.4, as well as the short comings of typical design approaches, as identified in Section 1.3. As a solution we proposed several contributions in the form a design framework for the specification and design exploration of complex FMSs for which we discuss the main conclusions in the remainder of this section.

7.1.1 Modular and Compositional Specification of FMSs

In Chapter 2 we introduced a formal and modular specification approach which is able to capture the flexibility characteristics of FMSs. In this approach an FMS is decomposed into a plant, activities and logistics. The plant abstracts the physical components of the system in terms of resources, peripherals and actions. On top of the plant, activities can be constructed to define certain functional behaviors of the system, such as transport or manufacturing operations. The complete manufacturing of a product, or a batch or products, is then captured by a specific ordering of multiple activities, which we call activity sequences. Because of the flexibility characteristics of these systems, such as the existence of multiple routing choices and multiple resource assignments, there can be multiple activity sequences encoding the correct and complete manufacturing of a product. This collection defines the logistics of the system which is encoded by a finite-state automaton, which we call a logistics automaton. Logistics automata can be defined modularly for each product in a batch. The resulting logistics automaton of a batch can be obtained via the composition of the individual logistics requirements for each product using a composition operator on logistics automata. 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 which can be composed with logistics automata using a constraining operator. The application of this specification approach was demonstrated by specifying the plant, activities and logistics of the Twilight

system introduced in Chapter 2.

The modularity enables the framework to deal with complex specifications by allowing different requirements to be specified in isolation. This specification framework guarantees that requirements specified as logistics and constraint automata are preserved by the composition and constraining operators. Further, the specification approach enforces separation of concerns by allowing behavior and timing aspects to be independently specified and analyzed. The abstractions and concepts used by the specification approach are based on those found in the design process of ASML system designers. Due to this, this approach has been integrated in a proprietary tool used within ASML for the design exploration of the logistics controllers of lithography scanners.

7.1.2 Design Exploration of FMSs

This thesis proposed two contributions in the design exploration of complex FMSs. In Chapter 3 we introduced an approach to automatically compute the optimal makespan of a given specification and in Chapter 5 we introduced a bottleneck identification approach. We discuss these in the following sections.

Makespan Optimization

In order to converge on a design specification for an FMS, it is important to be able to analyze and optimize its performance. In Chapter 3 we introduced the necessary concepts and methods such that different design specifications can be evaluated in terms of their expected makespan. Based on the formal specification approach introduced in Chapter 2 we defined the Batch Makespan Optimization (BMO) problem. A solution to the BMO problem can be obtained by finding the activity sequence with the lowest makespan within the language of activity sequences of the specified logistics automaton. In order to compute the makespan of an activity sequence, we introduced (max,+) semantics for activities where the temporal behavior of each activity is captured by a (max,+) matrix and an initial resource time-stamp vector. By the means of a (max,+) expansion algorithm we annotate the logistics automaton with the temporal characterization of activities to construct a (max,+) automaton. Each activity sequence in the language of the (max,+) automaton captures the manufacturing of a batch of products for which the completion time can be obtained by computing the resource time-stamp vector of the final states. Therefore the solution to the BMO problem can be obtained by exploring the state-space of the (max,+) automaton, which we call optimization-space.

This optimization technique allows for automatic minimization of the makespan of a given FMS specification. Therefore, allowing for the systematic exploration of design alternatives by comparing different design choices (e.g. different system layout layout or number of resources) in terms of their impact on the minimal makespan of a batch of products. This approach has been used to validate the specification of an industrial wafer handling controller and is also integrated in the a proprietary tool used for the design exploration of lithography scanners at ASML.

Identification of Performance Bottlenecks

To converge quicker on a final design specification one can use bottleneck identification techniques. These allow for a systematic approach to the identification of possible improvements to the design, instead of exploring different parameters by trial an error. In Chapter 5 we extended our framework with such a technique by introducing Stochastic Criticality Analysis (SCA). This analysis allows for the identification of performance bottlenecks in a sequenced activity, even when peripheral actions exhibit stochastic execution times. The SCA estimates the criticality of the actions of a sequenced activity by identifying how often certain actions appear on the critical path. Actions with high criticality point to improvement candidates (actions, peripheral and resources). We formalized the Stochastic Criticality Analysis (SCA) approach with formal mathematical support together with confidence intervals to obtain results with known accuracy and showed its application on the field of FMSs. We concluded that the SCA analysis provides more informative results in a single step than traditional critical path analysis commonly used for bottleneck design such as the Critical Path Method (CPM) or the Project Review and Evaluation Technique (PERT). As a consequence, fewer design iterations are expected to be required to converge to a final design that either satisfies performance requirements or for which no further improvements are possible.

7.1.3 An Algebra to Reason About State-space Sizes

In Chapter 3 we studied the complexity of the BMO problem and showed it to fall within the class of NP-Hard problems. This implies that finding optimal solutions to the BMO problem might take prohibitively long depending on the size of the optimization-space induced by the optimization problem. To allow the framework to scale towards systems of industrial complexity, we introduced in Chapter 4 an algebra of logistics automata to reason in a modular

(algebraic) way about (behavioural and structural) equivalence and inclusion relations between logistics automata. These allow a systematic relation of their languages, their state-space and optimization-space sizes and their solutions to the BMO problem. 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. In addition, we established sufficient conditions to ensure that the constraining of a logistics automaton leads to the pruning of its state-space and optimization-space. To this end, we introduced the notions of np-repulsiveness and p-attractiveness. 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. We showed that this method facilitates a specification style that systematically ensures the reduction of the state-space and optimization-space of a given system specification. 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). The use of both the specification style and heuristic approach is exemplified by solving the BMO problem for the Buffered Twilight system for different batch sizes.

7.1.4 Industrial Application

When developing this framework we targeted that these could be easily adopted by industrial practitioners. For this reason, we focused on developing solutions that were consistent with the industrial way-of-working and state-of-practice at ASML. In particular, by studying design approaches and the documentation and implementation of several wafer handling controllers. As a consequence, many concepts of the contributions of this thesis are inspired on design patterns and processes found at ASML. These include the abstractions and concepts chosen for the specification approach of Chapter 2, the inspiration for the formalization of over-specification of Chapter 4 and the bottleneck identification approach of Chapter 5. This effort led to many of the results in this thesis to be integrated in a proprietary tool, developed by ESI (TNO) in collaboration with ASML which is currently in active use by ASML architects to study the impact of different design choices for new and existing systems. An example of how this design framework can be used in an industrial context was shown in Chapter 6. In this chapter we used the design framework to specify a wafer handling controller, optimize the makespan of a FOUP and identify performance bottlenecks to improve the initial specification of the controller.

We showed that the modular specification of Chapter 2 is able to capture the flexibility characteristics of this industrial system and how modularity can be used to tackle its large specification. We showed how to compute a solution to the BMO of the specified wafer handling controller by using the algebra and specification style introduced in Chapter 4. Further, we formalized a non-essential requirement of the wafer handler 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. This reduction demonstrated the effectiveness of using over-specification as a means to prune the optimization-space of the logistics of flexible manufacturing systems.

7.2 Future Work

The work presented in this thesis does not solve all of the existing challenges in the specification and design exploration of flexible manufacturing systems. In this section we mention a few of the possible aspects of the work that remains to be done in both the specification and design exploration of FMSs but also discuss the step towards implementation.

Specification

- Often FMSs exhibit latency timing requirements concerning deadlines and release times. These emerge from disturbances from the physical and chemical processes involved (e.g. thermal temperature drift of a wafer) which are compensated for by enforcing more deterministic system timing (e.g. the just-in-time loading of a conditioned wafer on a chuck). The specification of these type of latency requirements has not been explored in the work presented in this thesis, but constitutes an important research direction to allow a broader class of flexible manufacturing systems to be specified.
- A flexible manufacturing system may exhibit multiple use-cases, such as different production flows and error scenarios. In this thesis we have only considered the specification of a single production use-case. It remains to be studied how to integrally specify the complete set of system use-cases in a modular way. Specially challenging is to find solutions to capture the interactions and transitions between different use-cases.
- The specification approach of this thesis does not consider possible feedback from the plant. For instance, to indicate the end location of

a resource after the execution of an activity or to simply state success or failure. Incorporation of feedback can be useful to concisely capture error scenarios and to identify logistics decision points (e.g. to transition between use-cases).

Design Exploration

- Currently our framework follows a two step approach to find the optimal makespan activity sequence of a given design specification. First the logistics automaton of a batch is obtained by composing all the individual product logistics automata and constraint automata. Then, the optimization-space is obtained via the (max,+) expansion of the composed logistics automaton and a solution is found by exploring the optimization-space. An interesting research 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 [72], could be applied in to avoid the exploration of redundant sequence.
- Another interesting research direction is the study and application of general heuristic-approaches to tackle optimization scalability as a complementary approach to that of Chapter 4. Of special interest would be to further study a sub-class of problem instances for which we restrict the expressive power to POMSETS and then generalize existing scheduling techniques with due-dates and deadlines to POMSETS with vectors of due-dates and deadlines.
- In this thesis we focused on the performance analysis of the system assuming and ASAP scheduling policy for activities. An interesting research direction would be to relax this assumption by allowing other scheduling policies and to allow for the analysis of the latency requirements mentioned in future work with respect to the specification.

Implementation and Applicability

- One of the shortcomings of typical design approaches is the strong inconsistency between specification and implementation. An interesting research question which is not investigated in this thesis, is the property-preserving synthesis of implementation artifacts (e.g. code) from a formal specification. This would eliminate inconsistencies be-

tween specification and implementation, provide more predictability and minimize implementation errors during the design process.

- In this thesis we defined an algebra to systematically relate the sizes of the state-space and optimizations-space between logistics automata. An interesting research direction would be the generalization of this algebra and its application towards finite-state automata.
- We have applied this approach to several academic case-studies, an industrial wafer handling controller as well as to the xCPS system in [12]. However, more case studies (e.g. in the printing domain) would help to investigate the limitations of the approach and to generalize it.

A | Twilight Specification

This appendix contains the specification of actions and activities of the Twilight system.

Table A.1: *Set of activities of the Twilight System.*

(**1**) LR_PickFromInput	(**5**) LR_PutOnDrill	(**9**) UR_PutOnCond
(**2**) LR_PutOnCond	(**6**) UR_PickFromDrill	(**10**) UR_PutOnOutput
(**3**) LR_PickFromCond	(**7**) UR_PickFromCond	(**11**) Condition
(**4**) LR_PickFromDrill	(**8**) UR_PutOnDrill	(**12**) Drill

Table A.2: *Twilight system actions.*

(l1) LR_mv_COND_to_IN	(u1) UR_mv_COND_to_DRILL	(d1) move_UP
(l2) LR_mv_IN_to_COND	(u2) UR_mv_DRILL_to_COND	(d2) move_DOWN
(l3) LR_mv_COND_to_DRILL	(u3) UR_mv_OUT_to_DRILL	(d3) drill_on
(l4) LR_mv_DRILL_to_COND	(u4) UR_mv_DRILL_to_OUT	(c1) condition
(l5) LR_mv_AT_to_ABOVE	(u5) UR_mv_AT_to_ABOVE	(cl1) clamp_on
(l6) LR_mv_ABOVE_to_AT	(u6) UR_mv_ABOVE_to_AT	(cl2) clamp_off

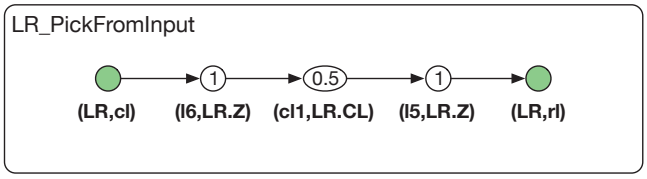


Figure A.1: *Activity LR_PickFromInput (**1**)*

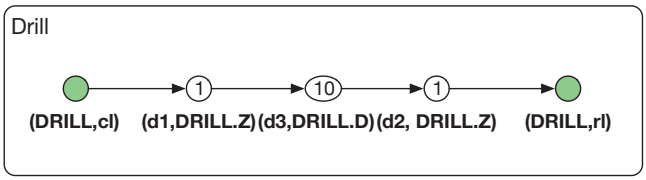


Figure A.2: *Activity Drill (**12**)*

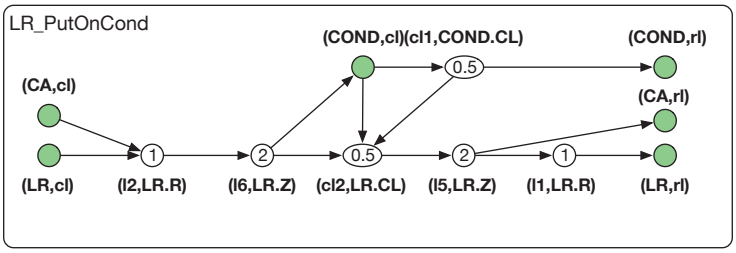


Figure A.3: *Activity LR_PutOnCond (**2**)*

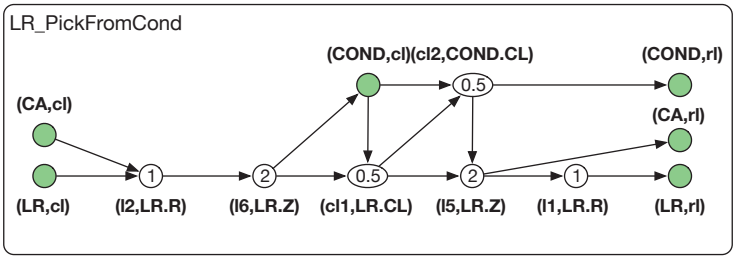


Figure A.4: *Activity LR_PickFromCond (**3**)*

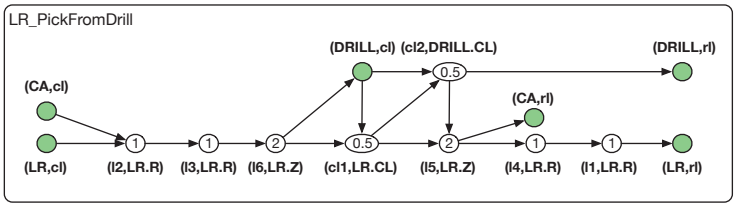


Figure A.5: *Activity LR_PickFromDrill (**4**)*

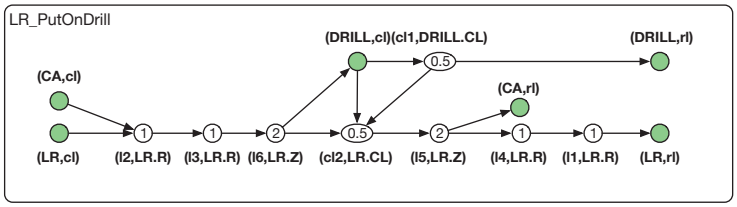


Figure A.6: *Activity LR_PutOnDrill (**5**)*

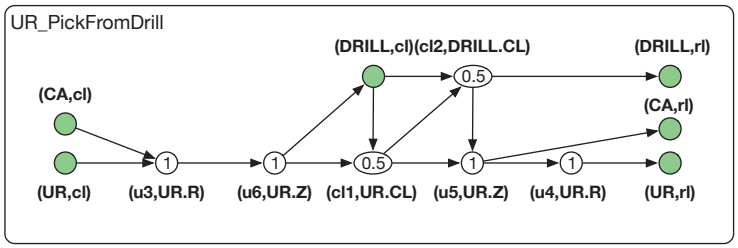


Figure A.7: *Activity LR_PickFromDrill (**6**)*

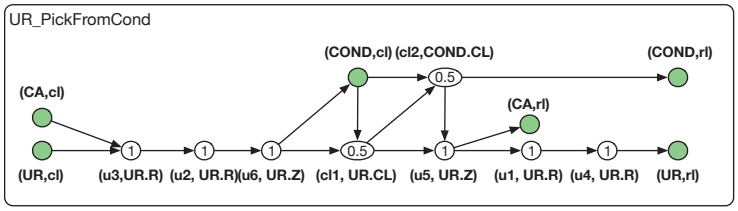


Figure A.8: *Activity LR_PickFromCond (**7**)*

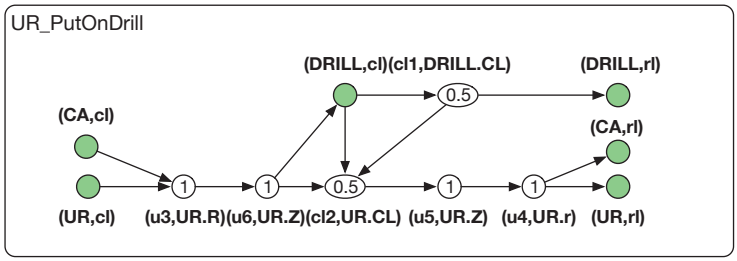


Figure A.9: *Activity LR_PutOnDrill (**8**)*

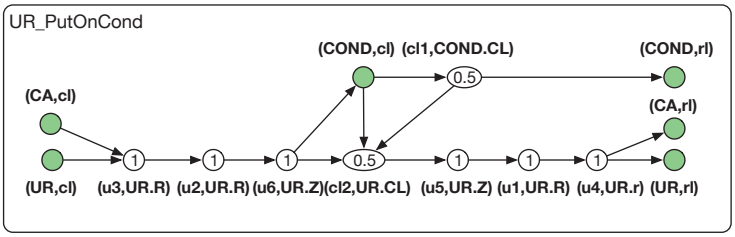


Figure A.10: *Activity LR_PutOnCond (**9**)*

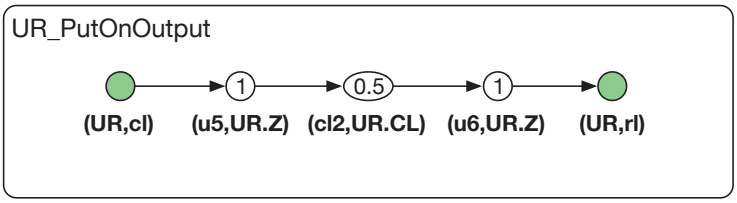


Figure A.11: *Activity LR_PutOnOutput (**10**)*

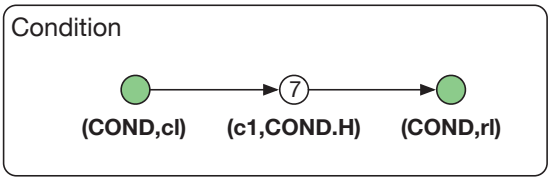


Figure A.12: *Activity Condition (**11**)*

B | Buffered Twilight Specification

This appendix contains the specification of actions and (additional) activities of the Buffered Twilight system.

Table B.1: *Set of activities of the Twilight Buffered system.*

(**1**) LR_PickFromInput	(**5**) LR_PutOnDrill	(**9**) UR_PutOnCond
(**2**) LR_PutOnCond	(**6**) UR_PickFromDrill	(**10**) UR_PutOnOutput
(**3**) LR_PickFromCond	(**7**) UR_PickFromCond	(**11**) Condition
(**4**) LR_PickFromDrill	(**8**) UR_PutOnDrill	(**12**) Drill
(**13**) LR_PickFromBuffer	(**14**) LR_PutOnBuffer	(**15**) UR_PickFromBuffer
(**16**) UR_PutOnBuffer	(**17**) DrillFast	

Table B.2: *Twilight Buffered system actions.*

(l1) LR_mv_COND_to_IN	(u1) UR_mv_COND_to_DRILL	(d1) move_UP
(l2) LR_mv_IN_to_COND	(u2) UR_mv_DRILL_to_COND	(d2) move_DOWN
(l3) LR_mv_COND_to_DRILL	(u3) UR_mv_OUT_to_DRILL	(d3) drill
(l4) LR_mv_DRILL_to_COND	(u4) UR_mv_DRILL_to_OUT	(c1) condition
(l5) LR_mv_AT_to_ABOVE	(u5) UR_mv_AT_to_ABOVE	(cl1) clamp_on
(l6) LR_mv_ABOVE_to_AT	(u6) UR_mv_ABOVE_to_AT	(cl2) clamp_off
(l7) LR_mv_IN_to_BUFFER	(u7) UR_mv_OUT_to_BUFFER	(d4) drill_fast
(l8) LR_mv_BUFFER_to_IN	(u8) UR_mv_BUFFER_to_OUT	

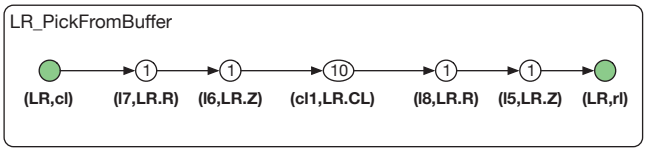


Figure B.1: *Activity LR_PickFromBuffer (**13**)*

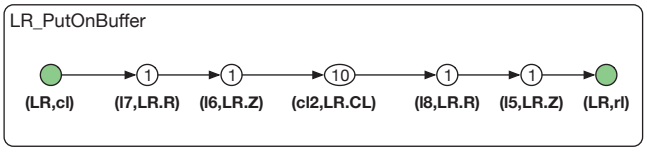


Figure B.2: *Activity LR_PutOnBuffer (**14**)*

Figure B.3: *Activity UR_PickFromBuffer (**15**)*

Figure B.4: *Activity UR_PutOnBuffer (**16**)*

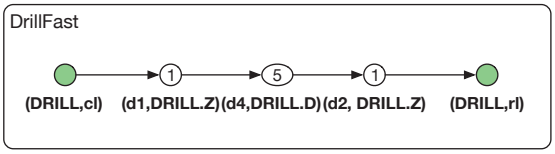


Figure B.5: *Activity DrillFast (**17**)*

Bibliography

[1] S. Adyanthaya, *Robust multiprocessor scheduling of industrial-scale mechatronic control systems*, PhD Thesis, Technische Universiteit Eindhoven, 2016.

[2] S. Adyanthaya, H. Alizadeh Ara, J. Bastos, A. Baghbanbehrouzian, R. Medina Sanchez, J. Pinxten, van, L. Sanden, van der, U. Waqas, A. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, S. Stuijk, M. Reniers, and J. Voeten, "xCPS : A tool to explore cyber physical systems", in *Proceedings of WESE'15 : Workshop on Embedded and Cyber-Physical Systems Education*, ACM, 2015, pp. 1–8.

[3] S. Adyanthaya, M. Geilen, T. Basten, R. Schiffelers, B. Theelen, and J. Voeten, "Fast multiprocessor scheduling with fixed task binding of large scale industrial cyber physical systems", in *2013 Euromicro Conference on Digital System Design (DSD)*, 2013, pp. 979–988.

[4] S. Adyanthaya, H. Alizadeh Ara, J. a. Bastos, A. Baghbanbehrouzian, R. M. Sánchez, J. van Pinxten, B. van der Sanden, U. Waqas, T. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, M. Hendriks, S. Stuijk, M. Reniers, and J. Voeten, "xCPS: A tool to explore cyber physical systems", *SIGBED Rev.*, vol. 14, no. 1, pp. 81–95, Jan. 2017.

[5] C. I. Ali and K. A. Ali, "A research survey: Review of flexible job shop scheduling techniques", *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.

[6] A. Allahverdi, "A survey of scheduling problems with no-wait in process", *European Journal of Operational Research*, vol. 255, no. 3, pp. 665 –686, 2016.

[7] A. Allahverdi, C. Ng, T. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs", *European Journal of Operational Research*, vol. 187, no. 3, pp. 985 –1032, 2008.

[8] R. Alur and D. L. Dill, "A theory of timed automata", *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[9] C. Andre, F. Mallet, and M. Peraldi-Frati, "A multiform time approach to real-time system modeling; application to an automotive system", in *2007 International Symposium on Industrial Embedded Systems (SIES)*, 2007, pp. 234–241.

[10] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity: An algebra for discrete event systems*, 2001.

[11] P. Barford and M. Crovella, "Critical path analysis of TCP transactions", *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 238–248, Jun. 2001.

[12] T. Basten, J. Bastos, M. Geilen, D. Goswami, R. Medina, M. Reniers, B. van der Sanden, S. Stuijk, and J. Voeten, "Scenarios in the design of flexible manufacturing systems", in *System Scenario-based Design Principles and Applications*, F. Catthoor, Ed., [under review], Springer, ch. 9.

[13] T. Basten, E. van Benthum, M. Geilen, M. Hendriks, F. Houben, G. Igna, F. Reckers, S. de Smet, L. Somers, E. Teeselink, N. Trčka, F. Vaandrager, J. Verriet, M. Voorhoeve, and Y. Yang, "Model-driven design-space exploration for embedded systems: The octopus toolset", in *Leveraging Applications of Formal Methods, Verification, and Validation*, T. Margaria and B. Steffen, Eds., Springer Berlin Heidelberg, 2010, pp. 90–105.

[14] J. Bastos, J. Voeten, S. Stuijk, H. Corporaal, and R. Schiffelers, "Exploiting specification modularity to prune the optimization-space of manufacturing systems", in *Exploiting Specification Modularity to Prune the Optimization-Space of Manufacturing Systems*, 2018.

[15] J. Bastos, S. Stuijk, J. Voeten, R. Schiffelers, J. Jacobs, and H. Corporaal, "Modeling resource sharing using fsm-sadf", in *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*. Sep. 2015, pp. 96–101.

[16] J. Bastos, B. van der Sanden, O. Donk, J. Voeten, S. Stuijk, R. Schiffelers, and H. Corporaal, "Identifying bottlenecks in manufacturing systems using stochastic criticality analysis", in *Proceedings of the 2017 Forum on Specification and Design Languages (FDL)*, vol. 2017-September, Feb. 2018, pp. 1–8.

[17] J. Bastos, J. Voeten, S. Stuijk, H. Corporaal, and R. Schiffelers, "Taming state-space explosion in the makespan optimization of flexible manufacturing systems", *ACM Transactions on Cyber-Physical Systems*, 2018, [under submission].

[18] D. A. van Beek, W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers, "CIF 3: Model-based engineering of supervisory controllers", in *TACAS*, 2014.

[19] E. Bertens, *Supervisory control synthesis for exception handling in printers*, Master Thesis, Technische Universiteit Eindhoven, 2009.

[20] P. Bjorn-Jorgensen and J. Madsen, "Critical path driven co-synthesis for heterogeneous target architectures", pp. 15–19, 1997.

[21] R. A. Bowman, "Efficient estimation of arc criticalities in stochastic activity networks", *Management Science*, vol. 41, no. 1, pp. 58–67, 1995.

[22] K. Braekers, K. Ramaekers, and I. V. Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review", *Computers and Industrial Engineering*, vol. 99, pp. 300 –313, 2016.

[23] M. A. van den Brink, H. Jasper, S. D. Slonaker, P. Wijnhoven, and F. Klaassen, "Step-and-scan and step-and-repeat: A technology comparison", *Proceedings of SPIE*, vol. 2726, pp. 2726 –2726 –20, 1996.

[24] E. Brinksma, "Constraint-oriented specification in a constructive formal description technique", J. W. de Bakker, W. P. de Roever, and G. Rozenberg, Eds., 1990.

[25] H. Butler, "Position control in lithographic equipment", *IEEE Control Systems*, vol. 31, no. 5, pp. 28–47, 2011.

[26] C. E. Clark, "Letter to the editor — the PERT model for the distribution of an activity time", *Operations Research*, vol. 10, no. 3, pp. 405–406, 1962.

[27] W. M. Cox and R. Alm, *You are what you spend*, https://www.nytimes.com/2008/02/10/opinion/10cox.html, 2008.

[28] S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, and T. A. C. Willemse, "An overview of the mCRL2 toolset and its recent advances", in *Tools and Algorithms for the Construction and Analysis of Systems*, N. Piterman and S. A. Smolka, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[29] B. M. Dodin and S. E. Elmaghraby, "Approximating the criticality indices of the activities in pert networks", *Management Science*, vol. 31, no. 2, pp. 207–223, 1985.

[30] R. Doering and Y. Nishi, "Handbook of semiconductor manufacturing technology", in *Microelectronics Journal*. Jul. 2007, vol. 32.

[31] K. Duisters, *Synthesis of supervisors for timed systems via region automata*, Master Thesis, Technische Universiteit Eindhoven, 2015.

[32] S. E. Elmaghraby, "On criticality and sensitivity in activity networks", *European Journal of Operational Research*, vol. 127, pp. 220–238, 2000.

[33] H. A. ElMaraghy, "Flexible and reconfigurable manufacturing systems parad igms", *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 261–276, 2005.

[34] eMarketer, "Smartphone penetration rate as share of the population in the united states from 2010 to 2021", *Statista*, 2018, https://www.statista.com/statistics/201183/forecast-of-smartphone-penetration-in-the-us/.

[35] R. Engelen, van and J. Voeten, *Ideals : Evolvability of software-intensive high-tech systems : A collaborative research project on maintaining complex embedded systems*. Technische Universiteit Eindhoven, Embedded Systems Institute, 2007.

[36] S. Forschelen, J. Mortel - Fronczak, van de, R. Su, and J. Rooda, "Application of supervisory control theory to theme park vehicles", *Discrete Event Dynamic Systems*, vol. 22, no. 4, pp. 511–540, 2012.

[37] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of np-completeness*. 1979.

[38] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios", in *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2010, pp. 125–134.

[39] M. Geilen, "Synchronous dataflow scenarios", *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, 16:1–16:31, Jan. 2011.

[40] R. de Groote, J. Kuper, H. Broersma, and G. J. M. Smit, "Max-plus algebraic throughput analysis of synchronous dataflow graphs", in *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, 2012, pp. 29–38.

[41] B. Heidergott, G. J. Olsder, and J. van der Woude, *Max plus at work: Modeling and analysis of synchronized systems: A course on max-plus algebra and its applications*. Princeton University Press, 2006.

[42] M. Hendriks, M. Geilen, A. R. B. Behrouzian, T. Basten, H. A. Ara, and D. Goswami, "Checking metric temporal logic with trace", in *16th International Conference on Application of Concurrency to System Design (ACSD)*, 2016.

[43] M. Hendriks, J. Verriet, T. Basten, M. Brassé, R. Dankers, R. Laan, A. Lint, H. Moneva, L. Somers, and M. Willekens, "Performance engineering for industrial embedded data-processing systems", in *Product-Focused Software Process Improvement*, Springer International Publishing, 2015, pp. 399–414.

[44] F. Hermans, M. Pinzger, and A. van Deursen, "Domain-specific languages in practice: A user study on the success factors", in *Model Driven Engineering Languages and Systems*, A. Schürr and B. Selic, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 423–437.

[45] J. Huang, J. Voeten, and H. Corporaal, "Predictable real-time software synthesis", *Real-Time Systems*, vol. 36, no. 3, pp. 159–198, 2007.

[46] R. M. Karp, "Reducibility among combinatorial problems", in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Springer, 1972.

[47] J. E. Kelley Jr and M. R. Walker, "Critical-path planning and scheduling", in *Eastern Joint IRE-AIEE-ACM Computer Conference*, ser. IRE-AIEE-ACM '59 (Eastern), New York, NY, USA, 1959, pp. 160–173.

[48] H. Kim, J. Lee, and T. Lee, "Time-Feasible Reachability Tree for Noncyclic Scheduling of Timed Petri Nets", *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 1007–1016,

[49] V. G. Kulkarni and V. G. Adlakha, "Markov and markov-regenerative pert networks", *Operations Research*, vol. 34, no. 5, pp. 769–781, 1986.

[50] E. A. Lee, M. Niknami, T. S. Nouidui, and M. Wetter, "Modeling and simulating cyber-physical systems using CyPhySim", in *2015 International Conference on Embedded Software (EMSOFT)*, 2015, pp. 115–124.

[51] H. Lei, K. Xing, L. Han, and Z. Gao, "Hybrid heuristic search approach for deadlock-free scheduling of flexible manufacturing systems using Petri nets", *Applied Soft Computing Journal*, vol. 55, pp. 413–423, 2017.

[52] L. Li, Q. Chang, J. Ni, G. Xiao, and S. Biller, "Bottleneck detection of manufacturing systems using data driven method", in *2007 IEEE International Symposium on Assembly and Manufacturing*, 2007, pp. 76–81.

[53] Y. Lu, K. Morris, and S. Frechette, "Current standards landscape for smart manufacturing systems", *National Institute of Standards and Technology*, Jan. 2016.

[54] K. R. MacCrimmon and C. A. Ryavec, "An analytical study of the pert assumptions", *Operations Research*, vol. 12, no. 1, pp. 16–37, 1964.

[55] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar, "Application of a technique for research and development program evaluation", *Operations Research*, vol. 7, no. 5, pp. 646–669, 1959.

[56] J. J. Martin, "Distribution of the time through a directed, acyclic network", *Operations Research*, vol. 13, no. 1, pp. 46–66, 1965.

[57] MathWorks, *Simulink toolbox: For use with matlab;* 2018.

[58] J. Mengerink, R. Schiffelers, A. Serebrenik, and M. van den Brand, "DSL/Model co-evolution in industrial EMF-based MDSE ecosystems", in *Proceedings of the 10th Workshop on Models and Evolution*, Oct. 2016, pp. 2–7.

[59] R. Milner, "Calculi for synchrony and asynchrony", *Theoretical Computer Science*, vol. 25, no. 3, pp. 267 –310, 1983.

[60] G. Moore, *Electronics Magazine*, no. 38, pp. 114 –117, 1965.

[61] T. Nishi and I. Matsumoto, "Petri net decomposition approach to deadlock-free and non-cyclic scheduling of dual-armed cluster tools", *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 1, pp. 281–294, 2015.

[62] J. van Pinxten, M. Geilen, T. Basten, U. Waqas, and L. Somers, "Online heuristic for the multi-objective generalized traveling salesman problem", in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 822–825.

[63] J. van Pinxten, U. Waqas, M. Geilen, T. Basten, and L. Somers, "Online scheduling of 2-re-entrant flexible manufacturing systems", *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 5s, 160:1–160:20, Sep. 2017.

[64] C. Ptolemaeus, Ed., *System design, modeling, and simulation using Ptolemy II*. Ptolemy.org, 2014.

[65] A. Rahatulain, "Modeling and Simulation of Evolvable Production Systems using Simulink / SimEvents",

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

[67] A. L. Ramos, J. V. Ferreira, and J. Barceló, "Model-based systems engineering: An emerging approach for modern systems", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 42, no. 1, pp. 101–111, 2012.

[68] A. Rensink and H. Wehrheim, "Weak sequential composition in process algebras", in *5th International Conference Proceedings (CONCUR '94)*, B. Jonsson and J. Parrow, Eds. Springer Berlin Heidelberg, 1994, pp. 226–241.

[69] J. G. Sánchez, M. M. Pascual, J. Marinero, F. Rojo, J. P. Turiel, and F. G. González, "Throughput Analysis of a Multirobot System Via Timed Petri Net Models", *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 247–252, 2002.

[70] B. van der Sanden, J. Bastos, J. Voeten, M. Geilen, M. Reniers, T. Basten, J. Jacobs, and R. Schiffelers, "Compositional specification of functionality and timing of manufacturing systems", in *2016 Forum on Specification and Design Languages (FDL)*, 2016, pp. 1–8.

[71] B. van der Sanden, M. Reniers, M. Geilen, T. Basten, J. Jacobs, J. Voeten, and R. Schiffelers, "Modular model-based supervisory controller design for wafer logistics in lithography machines", in *18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2015, pp. 416–425.

[72] B. van der Sanden, M. Geilen, M. A. Reniers, and T. Basten, "Partial-order reduction for performance analysis of max-plus timed systems", in *The 18th International Conference on Application of Concurrency to System Design (ACSD)*, 2018.

[73] R. R. H. Schiffelers, W. Alberts, and J. P. M. Voeten, "Model-based specification, analysis and synthesis of servo controllers for lithoscanners", in *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling (MPM)*, New York, NY, USA: ACM, 2012, pp. 55–60.

[74] O. Shmueli, N. Pliskin, and L. Fink, "Explaining over-requirement in software development projects: An experimental investigation of behavioral effects", *International Journal of Project Management*, vol. 33, no. 2, pp. 380 –394, 2015.

[75] C. Sigal, A. Pritsker, and J. Solberg, "The use of cutsets in monte carlo analysis of stochastic networks", *Mathematics and Computers in Simulation (MATCOM)*, vol. 21, no. 4, pp. 376–384, 1979.

[76] S. Stuijk, M. Geilen, and T. Basten, "SDF3: SDF For Free", in *Sixth International Conference on Application of Concurrency to System Design (ACSD'06)*, 2006, pp. 276–278.

[77] M. Teixeira, R. Lima, C. Oliveira, and P. Maciel, "A stochastic model for performance evaluation and bottleneck discovering on SOA-based systems", in *IEEE International Conference on Systems, Man and Cybernetics*, 2010, pp. 358–365.

[78] B. D. Theelen, O. Florescu, M. Geilen, J. Huang, P. H. A. van der Putten, and J. P. M. Voeten, "Software/hardware engineering with the parallel object-oriented specification language", in *Proceedings of the 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, IEEE Computer Society, 2007, pp. 139–148.

[79] B. Theelen and J. Hooman, "Uniting academic achievements on performance analysis with industrial needs", in *Quantitative Evaluation of Systems*, J. Campos and B. R. Haverkort, Eds., Springer International Publishing, 2015, pp. 3–18.

[80] R. Theunissen, M. Petreczky, R. R. H. Schiffelers, D. A. van Beek, and J. E. Rooda, "Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner", *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 1, pp. 20–32, 2014.

[81] R. Theunissen, "Supervisory control in health care systems", PhD thesis, Technische Universiteit Eindhoven, 2015.

[82] N. van den Nieuwelaar, "Supervisory machine control by predictive-reactive scheduling", PhD thesis, Technische Universiteit Eindhoven, 2004.

[83] R. M. Van Slyke, "Letter to the editor: Monte Carlo methods and the PERT problem", *Operational Research*, vol. 11, no. 5, pp. 839–860, Oct. 1963.

[84] U. Waqas, "Scheduling and variation-aware design of self-re-entrant flowshops", PhD thesis, 2017.

[85] U. Waqas, M. Geilen, J. Kandelaars, L. Somers, T. Basten, S. Stuijk, P. Vestjens, and H. Corporaal, "A re-entrant flowshop heuristic for online scheduling of the paper path in a large scale printer", in *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 573–578.

[86] M.-J. Yao and W.-M. Chu, "A new approximation algorithm for obtaining the probability distribution function for project completion time", *Computers and Mathematics with Applications*, vol. 54, no. 2, pp. 282 –295, 2007.

[87] Y. Yin, K. E. Stecke, and D. Li, "The evolution of production systems from industry 2.0 through industry 4.0", *International Journal of Production Research*, vol. 56, no. 1-2, pp. 848–861, 2018.

[88] Y. Zhan, A. J. Strojwas, M. Sharma, and D. Newmark, "Statistical critical path analysis considering correlations", in *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, IEEE Computer Society, 2005, pp. 699–704.

Publication List

- **Modeling resource sharing using FSM-SADF**, João Bastos, Jeroen Voeten, Sander Stuijk, Ramon Schiffelers, Johan Jacobs and Henk Corporaal, Proceedings of ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE), 21-23 September 2015, Austin, Texas.

- **Compositional specification of functionality and timing of manufacturing systems**, Bram van der Sanden, João Bastos, Jeroen Voeten, Marc Geilen, Michel Reiniers, Twan Basten, Johan Jacobs and Ramon Schiffelers, 2016, Proceedings of the 2016 Forum on specification and Design Languages, FDL 2016, Bremen, Germany, September 14-16, 2016.

- **Identifying bottlenecks in manufacturing systems using stochastic criticality analysis**, João Bastos, Bram van der Sander, Olaf Donk, Jeroen Voeten, Sander Stuijk and Henk Corporaal, Proceedings of the 2016 Forum on specification and Design Languages, FDL 2017, Verona, Italy, September 19-20, 2017.

- **Exploiting specification modularity to prune the optimisation-space of manufacturing systems**, João Bastos, Jeroen Voeten, Sander Stuijk and Henk Corporaal, Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems, SCOPES 2018, St. Goar, Germany, May 27-29, 2018.

- **xCPS : a tool to eXplore cyber physical systems**, S. Adyanthaya, H. Alizadeh Ara, João Bastos, A. Baghbanbehrouzian, R. A. Medina Sanchez, J.H.H. van Pinxten, Bram van der Sanden, U. Waqas, A.A. Basten, H. Corporaal, R.M.W. Frijns, M.C.W. Geilen, D. Goswami, S. Stuijk, M.A. Reniers

and J.P.M. Voeten, Proceedings of WESE'15 : Workshop on Embedded and Cyber-Physical Systems Education, 8 October 2015, Amsterdam, The Netherlands.

- **xCPS: a tool to explore cyber physical systems**, S. Adyanthaya, H. Alizadeh Ara, João Bastos, A. Baghbanbehrouzian, R. A. Medina Sanchez, J.H.H. van Pinxten, Bram van der Sanden, U. Waqas, A.A. Basten, H. Corporaal, R.M.W. Frijns, M.C.W. Geilen, D. Goswami, S. Stuijk, M.A. Reniers and J.P.M. Voeten, ACM SIGBED Review, 2016, Vol. 14, Pages 81-95.

- **Analyzing preemptive fixed priority scheduling of data flow graphs**, A. Lele, O. Moreira, C.H. van Berkel, João Bastos, R. Almeida and P. Pedreiras, Proceedings of the 12th Symposium on Embedded Systems for Real-time Multimedia ESTIMedia12, Greater Noida, India, October 16-17, 2014.

- **Mode-controlled data-flow modeling of real-time memory controllers**, Y. Li, H. Salunkhe, João Bastos, O. Moreira, B. Akesson and K.G.W. Goossens, Proceedings of the 13th IEEE Symposium on Embedded Systems for Real-Time Multimedia ESTIMedia13, Amsterdam, The Netherlands, October 8-9, 2015.

Acknowledgements

In these final pages I would like to leave a few words of gratitude to all of those that had a significant influence in this work and in my life as a PhD student.

First and foremost, I would like to express my deep gratitude to my first promotor Jeroen Voeten for all of his support and guidance throughout these years. His interest and excitement in my work have been a great source of motivation in the most difficult moments. His unique way of reasoning and understanding have been fundamental in helping me structure my thoughts and ideas, which eventually became the work in this thesis. I am also very thankful to him for sharing with me his insights and vision of applied research and how to better position my work in industry. It was a enormous pleasure to learn and grow with him as my mentor.

I would like to thank my second promotor Henk Corporaal for all his help and suggestions in my work. His rigorous critiques during our discussions have led to many interesting research directions and have made the work in this thesis more robust. I also appreciate his openness in sharing with me his vast experience as a researcher and as a teacher.

I would like to express my gratitude to my copromotor Sander Stuijk. His sharp and practical advices have helped me many times to see the simple in the complicated. He could quickly follow my reasoning and provide me with constructive feedback. I am also grateful for his patience and assistance in keeping me and my research on track.

Additionally, I would like to thank all of them for making it possible for me to finish this thesis remotely by ensuring that all the bureaucracy was timely filled and that the printed manuscripts were mailed to the correct places.

I would like to extend my gratitude to Bart Smolders, Jozef Hooman, Michel Reniers, Samarjit Chakraborty and Ramon Schiffelers for agreeing to review this manuscript and taking part in my doctoral committee.

I would especially like to thank Orlando Moreira for helping me take my first steps in research and advising me to pursue a PhD. Also, I am forever grateful to him for showing me that the Beatles are way more than just the Yellow Submarine.

During my time as PhD student I had the pleasure of being part of the Electronics Systems group. For that opportunity I would like to thank our group leader Twan Basten. I would also like to thank Marja for all the help and for her effort in creating a friendly working atmosphere. A special thanks to Andrew, Andreia, Dip, Francesco, Gabriela, Juan, Kees, Luc, Marc, Martijn, Mladen, Rasool, Rehan, Reinier, Ruben, Sajid, Shreya, Twan and Younghui for their help and friendship. An extended gratitude to Andrew for all the fun but also frustrating times we have spent around Computation II. Of course I could not forget the people from room 3.079 with whom I shared all the ups and downs of doing a PhD. Amir, Bram, Hadi, Joost, Robinson and Umar, thank you for being the noisy side of the room. Without your distractions I would have probably worked more but for sure not with the same joy. Bram, thank you for putting up with me all these years both at the TUe and at ASML. I have learned a lot from you and I will miss our coffee breaks and pointless distractions.

I would like to extend my gratitude to Johan Jacobs, Zef Alberti, Rainer Wolf and Ramon Schiffelers for their inputs in the project and for helping me understand the design and scheduling challenges faced by ASML. I would also like to thank Ramon and all the colleagues at the ASML Software Research Group for taking me in and creating a nice working environment. A special thank you to Yuri and Olaf for all the discussions and their contributions in the incorporation of our research results in the tool chain used within the ASML workflow. A warm thank you to Josh for all the Dutch and German cultural lessons and to my wining quiz mates Jurgen and Rolf (Rolf, the trophy is now yours to carry!).

One of my greatest pleasures outside of work has been the nights I have spent sitting around a table playing games. For this I thank my fellow Cthulhu cultists Orlando, Frauke, James, Celine and Michel for the many exciting and wonderful adventures we shared. Every game night was a big highlight of my week and something I will truly miss.

Being far away from my hometown was not always easy and I am very thankful to the friends I have made during these years in the Netherlands. Ivo, I cannot imagine Eindhoven without you. You welcomed me immediately as if we already new each other for years. I cherish all the long nights of

discussions and music discovery we have spent together. Luis, for me you will always be my one and only housemate. The times we spent at Meck5 are one of my fondest memories. Thank you for being my family abroad. Francesco, having had the chance to meet you made my life in the Netherlands feel more like home. All our metaphysical discussions helped me keep my mind sharp and healthy. Margarita, thank you for always listening to my problems and showing me how to be more positive. Nadine, thank you for enduring our stupid discussions and for providing a sense of style to our lives. Of course, thank you all for the parties and memorable nights we spent.

I would like to thank Frank and Jolanda for taking me into their home in a time of need and feeding and treating me like one of their own.

Despite the distance I could always count on the support of my friends in Portugal. You made it feel like not even a single day had passed by whenever I visited. A special thanks to Toni, Joana, Gi, Teresa, Maria, Tiago, Miguel, André for always making time for me, and to the six *palermas*, Teles, Miguel, Juta, Simão, Simões and Xina for still sticking with me after all these years.

To my parents, José and Emília. You have supported me through every step of my life and done so with so much love and affection. I thank you for always knowing what to say when I ask for advice and for standing behind me in every thing I do. *A vossa influência reflecte-se em tudo o que sou e em tudo o que faço.*

Thank you also to Kushi, my dear cat. Perhaps you did not help me like the others, but you did listen to me and to all my issues and never said anything back. Your silence and judgement kept me in line and your need for food and water gave me purpose when I could not find it elsewhere.

Finally, I would like to thank Immy. I am truly sorry you made the unthinkable decision of dating me when I was on the last years of my PhD. I should have warned you better that these were the worst ones! You have seen and suffered from all the stress and challenges alongside me. I am thankful for your patience and support in the good and (mostly) bad moments that came with it. Having you by my side gave me strength and tranquility when I needed it the most and I love you for that. I hope one day I can return the favor and I promise you (slightly) less stressful days from now on.

Curriculum Vitae

João Bastos was born in Porto, Portugal on December 23, 1989. He received his Master degree in Electronics and Telecommunications from the University of Aveiro, Portugal in 2013. He combined his Master thesis with an internship at Ericson, Eindhoven. His Master thesis focused on the scheduling and temporal analysis of hard real-time applications on multiprocessor platforms. In 2014 he joined the Electronic Systems group of Eindhoven University of Technology where he started his PhD within the Netherlands Organization for Scientific Research (NWO) rCPS (robust Cyber-Physical Systems) programme. This PhD project was a joint collaboration between the TU/e, ASML and ESI (TNO). During the course of his PhD, he devoted his time to the study and development of a model-based framework for the specification and design exploration of flexible manufacturing systems. The results of his PhD work have led to several publications and to the contents of this thesis. He currently works as an embedded software designer at Bosch, Portugal.