RWTH Aachen

Department of Computer Science
Technical Report

Automata and Logics for Message Sequence Charts

Benedikt Bollig

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2005-10

RWTH Aachen · Department of Computer Science · May 2005

The publications of the Department of Computer Science of RWTH Aachen (*Aachen University of Technology*) are in general accessible through the World Wide Web.

`http://aib.informatik.rwth-aachen.de/`

Automata and Logics for Message Sequence Charts

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Benedikt Bollig
aus
Düsseldorf

Berichter: Prof. Dr. Klaus Indermark
Prof. Dr. Wolfgang Thomas

Tag der mündlichen Prüfung: 07.03.2005

Abstract

A *message-passing automaton* is an abstract model for the implementation of a distributed system whose components communicate via message exchange and thereby define a collection of communication scenarios called message sequence charts. In this thesis, we study several variants of message-passing automata in a unifying framework. We classify their expressiveness in terms of state-space properties, synchronization behavior, and acceptance mode and also compare them to algebraic characterizations of sets of message sequence charts, among them the classes of recognizable and rational languages.

We then focus on finite-state devices with global acceptance condition that communicate via a priori unbounded channels. We show them to be exactly as expressive as the existential fragment of monadic second-order logic over message sequence charts and to be strictly weaker than full monadic second-order logic. It turns out that message-passing automata cannot be complemented and that they cannot be determinized in general. Those results rely on a new proof technique, which allows to apply graph acceptors as introduced by Thomas to the framework of message sequence charts.

Zusammenfassung

Ein kommunizierender Automat ist ein abstraktes Modell eines verteilten Systems. Er besteht aus einigen Automaten, die über unbeschränkte FIFO-Kanäle Nachrichten austauschen und auf diese Weise eine Menge von Kommunikationsszenarien bzw. Message Sequence Charts (MSCs) definieren. In dieser Arbeit klassifizieren wir kommunizierende Automaten nach ihrem Zustandsraum, ihrer Akzeptanzbedingung und der Anzahl sogenannter Synchronisationsnachrichten und diskutieren die Ausdrucksstärke solcher Varianten bzgl. anderer Formalismen, etwa bzgl. der Klassen erkennbarer und rationaler Sprachen.

Im weiteren Verlauf der Arbeit konzentrieren wir uns auf kommunizierende Automaten mit endlichem Zustandsraum. Wir zeigen mit Hilfe der von Thomas eingeführten Graph-Akzeptoren, die bereits logisch charakterisiert wurden, dass endliche kommunizierende Automaten gerade diejenigen MSC-Sprachen erkennen, die im existentiellen Fragment der monadischen Logik zweiter Stufe, interpretiert über MSCs, definierbar sind, und dass die volle monadische Logik zweiter Stufe über MSCs ausdrucksstärker ist als ihr existentielles Fragment. Diese logische Charakterisierung lässt nun den Schluss zu, dass endliche kommunizierende Automaten im Allgemeinen nicht komplementierbar sind und dass nicht alle endlichen kommunizierenden Automaten in einen äquivalenten deterministischen Automaten transformiert werden können.

Acknowledgments

My foremost thank goes to my advisor Prof. Dr. Klaus Indermark. I thank him for his patience and encouragement, his advice and suggestions, which have influenced me greatly as a computer scientist and helped to shape my research skills.

I am grateful to Prof. Dr. Wolfgang Thomas for being a member of my thesis committee. Without his fundamental research, the results of this thesis would not have been possible.

I thank Prof. Dr. Marta Kwiatkowska for giving me the opportunity to work with her and her excellent group. I am grateful to the German Academic Exchange Service (DAAD) for providing financial support during this memorable time. Not only did I benefit from the research at Birmingham University, I also made friends there: Marie, Wang, Dave, Dimitar, and Steve (in order of appearance) have been much more than a "lunch group" to me.

My special thanks go to Dr. Martin Leucker, who introduced me to research, message sequence charts, and so much more. I learned a lot from his knowledge and it has been a privilege working with him and having him as a friend.

I thank my mate Darius for being a great friend and for always listening to me no matter what I have to say.

Thanks are given to everybody at I2 for a very stimulating and friendly research environment, which contributed a lot to this thesis.

Last, but certainly not least, let me record my sincerest thanks to Kerstin for all her love, care, and patience, and to my parents and to my sister for the unfailing support they have provided me with throughout.

Benedikt Bollig
Aachen, May 2005

Contents

Chapter 1

Introduction

Nowadays, electronic devices largely depend on complex hardware and software systems. Among them, medical instruments, traffic control units, and many more safety critical systems are subject to particular quality standards. They all come along with the absolute need for reliability, as, in each case, the consequences of a breakdown may be incalculable.

1.1 Formal Methods

Many existing systems have been unthinkable some years ago and their complexity is still rapidly growing. Consequently, it becomes more and more difficult to detect errors or to predict their incidence. *Formal methods* play a major role during the whole system-design process. The term formal methods hereby covers a wide range of mathematically-derived and ideally mechanized approaches to system design and validation. More precisely, research on formal methods attempts to develop mathematical models and algorithms that may contribute to the tasks of *modeling*, *specifying*, and *verifying* software and hardware systems. Let us go into these subareas in more detail:

Modeling To make a system (or the idea of a system) accessible to formal methods, we require it to be modeled mathematically. Unfortunately, this directly leads into a dilemma: on the one hand, a model ideally preserves and reflects as many properties of the underlying system as possible. On the other hand, it should be compact enough to support algorithms for further system analysis. However, in general, a good balance between a detailed modeling and abstraction will pay off. But not only does the modeling process lead to further interesting conclusions, it may also help itself to get a better understanding of the system at hand. Thus, the purposes of modeling a system are twofold. One is to understand and document its essential features. The other is to provide the formal basis for a mathematical analysis. Both are closely related and usually accompanied by each other. Preferably, the modeling takes place in an early stage of system design. The starting point, at a high level of abstraction, may be a rough, even if precisely defined, idea of the system to be, which is subsequently refined stepwise towards a full implementation. While, as mentioned, the latter might be too detailed to draw conclusions from, previous stages of the design phase can be consulted for that purpose.

The models considered in this thesis are *message-passing automata*, which, though they might abstract from many details, reflect the behavior of a distributed system in a suitable manner to make it accessible to formal methods.

Specification Correctness of a system is always relative to a *specification*, a property or requirement that is essential to be satisfied. Embedded into the formal-methods framework, a specification is often expressed within a logical calculus whose formulas can be interpreted over system models, provided they are based on a common semantic domain. Prominent examples are monadic second-order logic [HJJ$^+$95, ABP97], the temporal logics LTL [Pnu77], CTL [CE81], and the μ-calculus [Koz83]. A specification might also be given in a high-level language that is closer to an implementation and often allows to derive a system directly and automatically. In this regard, let us mention some process-algebra based languages such as CCS [Mil89], ACP [BV94], and LOTOS [BB89] and other formal design notions like VHDL [Per91] and UML [SP99].

In this thesis, which is about the design of communicating systems, we focus on a monadic second-order logic, which might be used to formulate properties that a *given* system should satisfy, and high-level message sequence charts, which are employed at a rather early stage of system development.

Verification Once a system is modeled and a specification is given, the next task might be to check if the specification is satisfied by the model. Thus, if the system or, rather, the model of a system passes successfully through a corresponding validation process, it may be called correct in a mathematical sense. Preferably, the verification process runs automatically. However, many frameworks are too complex to support fully mechanized algorithms. In this respect, we can distinguish two general approaches to verification: *model checking* [CGP99], which is fully automatic, and *theorem proving* [RV01], which requires human assistance. Another approach is to derive a system directly from its specification so that, provided the translation preserves the semantics of the specification, it can be assumed to be correct a priori.

In this thesis, we rather concentrate on the latter approach and point out some limitations that come along with the high-level specification languages mentioned in the previous section.

1.2 Modeling Message-Passing Systems

To create formal methods tailored to a given kind of system and the associated mathematical model, it is generally helpful to study some of the model's properties first and to learn more about its limitations coming along with algorithmic restrictions and its degree of abstraction. In this regard, typical questions to clarify are "Is my model a suitable one, i.e., does it reflect all the aspects I like to verify?" or "What kind of problem can I expect to be decidable at all?". Basically, that is what this thesis is all about. More specifically, we will concentrate on communicating systems (or message-passing systems), which occur whenever processes and objects communicate or interact, for example, via message exchange. At the same time, we focus on issues related to the former two areas of formal methods, i.e., system modeling and specification. However, the study of modeling and specification languages interferes with finding algorithms for their verification. For instance, one particular model of a message-passing system will be implicitly shown to be too complex (though it is a common and natural one) to be applicable to verification procedures.

1.2.1 Message Sequence Charts

As mentioned above, it is desirable to apply formal methods even in the early stages of system design, as early error detection avoids extensive reimplementation and redesign, which, in turn, might lead to explosive costs. A common design practice when developing communicating systems is to start with drawing scenarios showing the intended interaction of the system to be. *Message Sequence Charts* (MSCs), a modeling language at a rather high level of abstraction, provide a prominent notion to further this approach. They are widely used in industry, standardized [ITU98, ITU99], and similar to UML's sequence diagrams [Ara98]. An MSC depicts a single partially ordered execution sequence of a system. In doing so, it defines a collection of processes, which, in its visual representation, are drawn as vertical lines and interpreted as time axes. Moreover, an arrow from one line to a second corresponds to the communication events of sending and receiving a message.

An example MSC illustrating a part of *Bluetooth^{TM}* [Blu01], a specification for wireless communication, is depicted in Figure 1.1 on the facing page. Using the *Host Control Interface* (HCI), which links a Bluetooth host (a portable PC, for example) with a Bluetooth controller (a PCMCIA card, for example), a host application attempts to establish a connection to another device. The connection-request phase, which is based on an asynchronous connectionless link (ACL), is heralded by Host-A sending an HCI_Create_Connection command to its controller to initiate a connection. Note that, usually, a command is equipped with parameters, which are omitted here. As HCI commands may take different amounts of time, their status is reported back to the host in form of an HCI_Command_Status event. After that, HC-A defers the present request to HC-B, which, in turn, learns from Host-B that the request has been rejected, again accompanied by sending a status event. The controllers agree on rejection by exchanging messages LMP_not_accepted and LMP_detach and, afterwards, provide both Host-A and Host-B with HCI_Connection_Complete events.

The execution sequence illustrated above is geared to the visual arrangement of the message arrows' endpoints. An endpoint is a *send event* if it belongs to the source of some arrow and, otherwise, a *receive event*. More specifically, we suppose events located on one and the same process line to be totally ordered and, moreover, require a receiving event to occur only if the corresponding sending event has taken place. The abovementioned partial order now arises from the reflexive transitive closure of those assumptions. Note that, in fact, some pairs of events cannot be ordered accordingly. Considering our example, receiving the HCI_Command_Status event by Host-A may occur before or after receiving LMP_host_connection_req, while the latter is supposed to happen after sending the former HCI_Command_Status event.

1.2.2 MSC Languages

Recall that a specification language might be used to formulate desirable properties of a given implementation or represent a first intuition of what the system has to do. While a single MSC can hereby describe no more than one single execution sequence, a collection of MSCs might capture all the scenarios that a designer wants the system under development to realize. Based on the notion of MSCs, several modeling and specification formalisms have been considered at a formal level, among them *high-level MSCs* [MR97, AY99, MP99, HMKT00a], which are capable of describing possibly infinitely many scenarios in a compact manner. From an algebraic point of view, high-level MSCs are rational expressions defining *rational MSC languages* by means of choice, concatenation, and

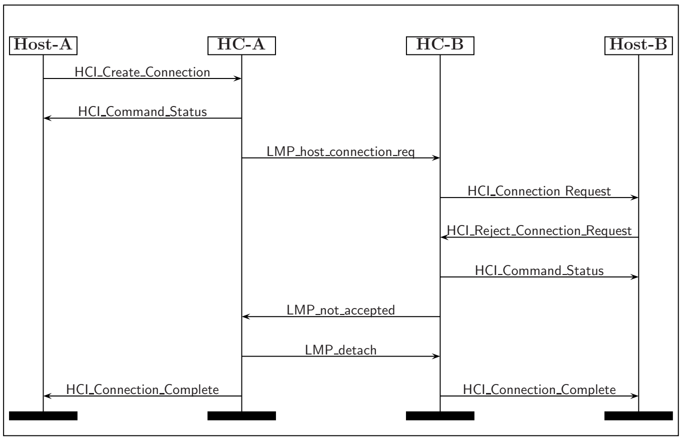


Figure 1.1: An MSC modeling the ACL connection request phase

iteration. The study of algebraic language classes might then lead us to *recognizable MSC languages* [Mor02], which can be characterized by certain *monoid automata*. Following the classical algebraic approach further, we will come across the class of *regular MSC languages*. In their seminal work, Henriksen et al. consider an MSC language to be regular if the corresponding set of *linear extensions* forms a regular word language [HMKT00b]. Moreover, there is a close connection between MSCs and Mazurkiewicz traces so that transferring the regularity notions for traces might be another axis to define regularity of sets of MSCs. Those aspects have been studied in [Mor01, Kus02, Mor02, GKM04]. As we will see, the above language classes exhibit quite different properties in terms of *implementability*. Hereby, the notion of implementability is derived from a reference model, the MSC based counterpart of a finite automaton over words, which is explained in the next section in more detail.

All times, a fruitful way to study properties of language classes has been to establish their logical characterizations. The study of formalisms for general structures such as graphs or, more specifically, labeled partial orders and their relation to monadic second-order (MSO) logic has been a research area of great interest aiming at a deeper understanding of their logical and algorithmic properties (see [Tho97b] for an overview). Following the logical approach, one might likewise argue to call a set of MSCs regular if it is definable in the MSO logic adjusted to MSCs, because, in the domain of words, regularity and definability in MSO logic coincide [Büc60, Elg61]. This view was also pursued in [HMKT00b] where it is in fact shown that MSO captures the class of regular languages when formulas are supposed to define *bounded* sets of MSCs. Intuitively, an MSC language is bounded if its structure exhibits a bound on the number of unreceived messages. However, unlike corresponding logics over words and traces, MSO logic interpreted over arbitrary MSCs turns out to be too expressive to be considered as a reference logic, as formulas may describe languages that are not implementable. However, we prove its *existential* fragment to capture exactly the implementable specifications and therefore introduce the class of *EMSO-definable languages*, which lifts the boundedness restriction without abandoning the existence of an automata theoretic counterpart. More precisely, we show that an MSC language is implementable if and only if it is definable in EMSO logic.

1.2.3 Towards an Implementation

The next step in system design might be to supply an implementation realizing or satisfying an MSC specification. Recall that we are still interested in an abstract model rather than a concrete implementation in some low-level programming language, though the view we are taking now might be much closer to the latter. More precisely, we ask for *automata* models that are suited for accepting the system behavior described by, say, a high-level MSC, a logical formula, or a monoid automaton. Consequently, we are particularly interested in their expressiveness relative to the abovementioned algebraic characterizations.

MSCs exemplify systems that are distributed in nature. Thus, the notion of a process is central, and it seems natural to consider each process as a single automaton and to define a notion of communication describing how these parallel systems work together. Such a collection of communicating automata gives rise to a *message-passing automaton* (MPA). Typically, automata are equipped with some acceptance mode. We discuss MPAs with two different notions of an acceptance condition, which is either *local*, i.e., each process decides on its own when to stop, or it is *global*, which allows to select certain combinations of local final states to be a valid point for termination. It is not surprising that, provided the system has a single initial state, the latter notion of acceptance is generally more expressive than the local one. Expressiveness can also be considerably increased providing *synchronization messages* sent in addition to the actual message contents. Intuitively, a synchronization message can inform other components about which transition was taken. We will show that, the more of these messages are allowed, the more expressive power we have.

Note that, so far, we did not impose any restriction on the cardinality of a local state space. In the course of our examination, however, *finite* MPAs, which are equipped with a global acceptance mode, a device to send synchronization messages, and a finite state space for each process, will emerge to be kind of reference model, which represents what we subsequently call implementable. We will show that, to some extent, finite MPAs are to MSCs as *asynchronous automata* are to traces and finite automata are to words. They are more powerful than standard models considered in previous work, which mainly concentrates on bounded or unsynchronized behavior. Though finite MPAs employ finite local state spaces, they are very well suited to generating an unbounded behavior and, in particular, allow for the recognition of any regular MSC language.

1.3 Contribution and Outline of this Thesis

Even if MPAs are subject to active research, there is no agreement which automata model is the right one to make an MSC language *realizable*: while, for example, [GMSZ02] is based on a local acceptance condition and allows to send synchronization messages, [AEY00, Mor02] forbid those extra messages. In [HMKT00b], though a priori unbounded channels are allowed, a channel-bounded model, equipped with a global acceptance mode, suffices to implement regular MSC languages. We provide a unifying framework and classify automata models in terms of their state space, synchronization behavior, acceptance mode, and, on a rather semantical level, whether they generate bounded or unbounded behavior. We then focus on finite MPAs with a global acceptance condition that communicate via a priori unbounded channels. While previous work lacks an algebraic or logical characterization of the corresponding class of languages, we show those automata to be exactly as expressive as EMSO and to be strictly less expressive than MSO logic. Thus, we provide a specification formalism that captures all the implementable MSC languages and, at the same time, highlight its limitations. As a by-product, we show that finite MPAs cannot be complemented in general and, in doing so, reveal a new proof technique applying Thomas' *graph acceptors* to MSCs. Moreover, we generalize existing results comparing finite MPAs to high-level MSCs, to recognizable MSC languages, and to some variants of MSO logic. Summarizing, we separate specification formalisms that yield implementable languages from formalisms that are incompatible with the reference model of an implementation.

The next chapter recalls fundamental notions and results that serve as a basis for the study of MSCs. MSCs are introduced formally in Chapter 3, followed by the definition of several classes of MSC languages. The definition of an MPA can be found in Chapter 4, which also deals with its expressiveness relative to the previously proposed language classes. Finally, Chapter 5 studies the gap between MSO logic and its existential fragment, which is also compared to the formalisms of high-level MSCs and monoid automata.

Some results presented in this thesis appear in [BL04, BL05].

Chapter 2

Graphs, Words, Traces, and Pictures

In this chapter, we study classes of structures that might be appropriate to describe and model the behavior of a distributed system. We hereby come from quite general structures, which, depending on the kind of system at hand, are refined towards more specific ones. Our starting point will be the class of (directed) graphs that generate partial orders in a natural manner. The nodes of a graph can then be seen as events, which are executed in the order imposed by the edge relation. They are partially ordered rather than queued in a total order to abstract from a concrete ordering of intrinsically independent events. However, arbitrary partial orders or, rather, their associated graphs might be too general to model behaviors with special characteristics. For example, if we deal with finite automata as a model of a system, it might suffice to consider only those graphs that represent totally ordered sets or words. Moreover, if the system at hand is designed for sending messages between finitely many processes each of which is sequential, only those graphs come into question whose edges reflect either a message exchange or neighboring events executed by one and the same sequential process. As the distributed behavior of a concurrent system gives rise to events that cannot be ordered, we shall furthermore exclude graphs provoking such an unnatural ordering of events. While the former model, a message-passing system, will lead us to *message sequence charts*, the latter refers to *Mazurkiewicz traces* [DR95]. As we will see, message sequence charts and Mazurkiewicz traces are incomparable in general, but, in some special cases, can be embedded into each other.

Besides models for executions of a distributed system, we also study models for the system itself. A distributed system is represented by an automaton, which generates a set of behaviors by processing events along the partial order in a transition-based manner. Finite automata provide the basic automata model, which works over words reading them sequentially and letter by letter. Finite automata can be combined towards more complex systems: several components are henceforth able to communicate with one another. For example, in a *message-passing system*, a finite automaton can execute send actions and, in doing so, write messages in a buffer. Those messages can be read by another automaton whose further behavior depends on the message contents. Another type of communication is used by *asynchronous automata* where some transitions are taken autonomously, while others can only be executed if another component agrees.

2.1 Partial Orders and Monoids

As mentioned above, one possible single behavior of a distributed system can be described in a compact manner by a partially ordered set. In this regard, let us first recall some basic definitions and notions.

A binary relation $\leq \subseteq E \times E$ on a set E is called

- *reflexive* if, for each $e \in E$, $e \leq e$,
- *transitive* if, for any $e, e', e'' \in E$, $(e \leq e' \wedge e' \leq e'')$ implies $e \leq e''$, and
- *antisymmetric* if, for any $e, e' \in E$, $(e \leq e' \wedge e' \leq e)$ implies $e = e'$.

Definition 2.1.1 (Partially Ordered Set)
A *(finite) partially ordered set* (also called *poset*) is a pair $(E, \leq)$ such that

- E is a finite set and
- $\leq$ is a binary relation on E that is reflexive, transitive, and antisymmetric.

In this context, the relation $\leq$ is called a *partial order*.

A *totally ordered set* is a poset $(E, \leq)$ such that, for any $e, e' \in E$, $e \leq e'$ or $e' \leq e$. Accordingly, we then call the relation $\leq$ a *total order*. Note that, throughout this thesis, we do not distinguish isomorphic structures.

Let $\mathcal{P} = (E, \leq)$ be a poset. By $<$, we denote the binary relation $\leq \setminus \{(e, e) \mid e \in E\}$. Moreover, for $e, e' \in E$, let us write $e \lessdot e'$ if both $e < e'$ and, for any $e'' \in E$, $e < e'' \leq e'$ implies $e'' = e'$. Then, $(E, \lessdot)$ and $\lessdot$ are called the *Hasse diagram* of $\mathcal{P}$ and, respectively, the *covering relation* of $\leq$. For $e \in E$, we furthermore say that e is *minimal*/*maximal* in $\mathcal{P}$ (we also say minimal/maximal in $(E, <)$) if there is no $e' \in E$ such that $e' < e$/$e < e'$, respectively.

The structures considered in this thesis are often equipped with a concatenation function, which allows to combine single behaviors towards more complex ones. Together with a unit element, this gives rise to a monoid.

Definition 2.1.2 (Monoid)
A *monoid* is a triple $(\mathbb{M}, \cdot, \mathbf{1})$ such that $\mathbb{M}$ is a nonempty set, $\cdot$ is an associative mapping $\mathbb{M} \times \mathbb{M} \to \mathbb{M}$ (i.e., $(t_1 \cdot t_2) \cdot t_3 = t_1 \cdot (t_2 \cdot t_3)$ for any $t_1, t_2, t_3 \in \mathbb{M}$), and $\mathbf{1} \in \mathbb{M}$ is the *unit* satisfying $\mathbf{1} \cdot t = t \cdot \mathbf{1} = t$ for any $t \in \mathbb{M}$.

A monoid $(\mathbb{M}, \cdot, \mathbf{1})$ is often identified with $\mathbb{M}$.

Let $(\mathbb{M}, \cdot, \mathbf{1})$ be a monoid. A subset of $\mathbb{M}$ is called a *language*. Given languages $L, L' \subseteq \mathbb{M}$, the *product* of L and L' is denoted by $L \cdot L'$ and defined to be the set $\{t \cdot t' \mid t \in L \text{ and } t' \in L'\}$. Furthermore, we set $L^0 = \{\mathbf{1}\}$ and, for $n \in \mathbb{N}$ (where $\mathbb{N}$ denotes the set of naturals including 0), $L^{n+1} = L \cdot L^n$. The *iteration* of L is defined to be $L^* := \bigcup_{n \in \mathbb{N}} L^n$, which is also denoted by $\langle L \rangle_{\mathbb{M}}$. Note that $\langle L \rangle_{\mathbb{M}}$ is a submonoid of $\mathbb{M}$. By L^+, we abbreviate $\bigcup_{n \in \mathbb{N}_{\geq 1}} L^n$ where $\mathbb{N}_{\geq 1}$ will throughout this thesis stand for the set of positive natural numbers. A language $L \subseteq \mathbb{M}$ is called *finitely generated* if there is a finite subset Γ of $\mathbb{M}$ such that $L \subseteq \langle \Gamma \rangle_{\mathbb{M}}$. In that case, we say that L is finitely generated by Γ.

The class $\mathcal{RAT}_{\mathbb{M}}$ of *rational* subsets of $\mathbb{M}$ is the least set satisfying

- $\emptyset \in \mathcal{RAT}_{\mathbb{M}}$,
- $\{t\} \in \mathcal{RAT}_{\mathbb{M}}$ for any $t \in \mathbb{M}$, and
- for $L_1, L_2, L \in \mathcal{RAT}_{\mathbb{M}}$, $L_1 \cdot L_2$, $L_1 \cup L_2$, and L^* are contained in $\mathcal{RAT}_{\mathbb{M}}$.

In other words, a rational language can be obtained from the finite subsets of $\mathbb{M}$ by means of finitely many unions, products, and iterations, which gives rise to a *rational expression* of $\mathbb{M}$. Formally, the set of atomic rational expressions is $\mathbb{M} \uplus \{\emptyset\}$, which can be combined towards expressions $\alpha_1 \cdot \alpha_2$, $\alpha_1 + \alpha_2$, and α^* (α^+ will henceforth stand for $\alpha \cdot \alpha^*$). We denote by $L(\alpha)$ the language that corresponds to the rational expression α of $\mathbb{M}$, i.e., $L(\emptyset) = \emptyset$, $L(t) = \{t\}$ for any $t \in \mathbb{M}$, $L(\alpha_1 \cdot \alpha_2) = L(\alpha_1) \cdot L(\alpha_2)$, $L(\alpha_1 + \alpha_2) = L(\alpha_1) \cup L(\alpha_2)$, and $L(\beta^*) = L(\beta)^*$. Obviously, any rational language is finitely generated.

A subset L of $\mathbb{M}$ is called *recognizable* if there exists a finite monoid $(\mathbb{M}', \cdot', \mathbf{1}')$ and a monoid morphism $\eta : \mathbb{M} \to \mathbb{M}'$ (i.e., $\eta(t_1 \cdot t_2) = \eta(t_1) \cdot' \eta(t_2)$ for any $t_1, t_2 \in \mathbb{M}$ and $\eta(\mathbf{1}) = \mathbf{1}'$) such that $L = (\eta^{-1} \circ \eta)(L)$. Note that the set of recognizable subsets of $\mathbb{M}$ is closed under union, intersection, and complement. Recognizability can be defined equivalently in terms of *monoid automata*. Formally, an $\mathbb{M}$-*automaton* is a quadruple (Q, δ, q_0, F) where Q is the nonempty finite set of *states*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, and δ is a function $Q \times \mathbb{M} \to Q$ such that, for any $q \in Q$ and $t_1, t_2 \in \mathbb{M}$, $\delta(q, \mathbf{1}) = q$ and $\delta(\delta(q, t_1), t_2) = \delta(q, t_1 \cdot t_2)$, which can be considered to be some compositional rule. We might now call a language $L \subseteq \mathbb{M}$ recognizable if $L = \{t \in \mathbb{M} \mid \delta(q_0, t) \in F\}$ for some $\mathbb{M}$-automaton (Q, δ, q_0, F). The set of recognizable subsets of $\mathbb{M}$ is denoted by $\mathcal{REC}_{\mathbb{M}}$. The following is well-known.

Proposition 2.1.3 For any monoid $\mathbb{M}$, $\mathbb{M} \in \mathcal{RAT}_{\mathbb{M}}$ iff $\mathbb{M}$ is finitely generated.

Proposition 2.1.4 For any monoid $\mathbb{M}$, $\mathcal{REC}_{\mathbb{M}} \subseteq \mathcal{RAT}_{\mathbb{M}}$ iff $\mathbb{M}$ is finitely generated.

See [Och95] and [Cou90] for comprehensive overviews of recognizability and rationality with a special focus on traces and graphs, respectively.

2.2 Graphs and Graph Acceptors

Directed acyclic labeled graphs can be seen as the most general structure we consider in this thesis. Many structures that will also be addressed, such as words and (graphs associated to) partial orders, can be embedded into acyclic graphs or at least have a corresponding one-to-one graph representation.

2.2.1 Graphs

Let in the following Σ and C be *alphabets*, i.e., nonempty finite sets, which contain the elements the components of a graph are labeled with.

Definition 2.2.1 ((Directed) Graph)
A *(finite, directed) graph* over (Σ, C) is a structure $(E, \{\lhd_c\}_{c \in C}, \lambda)$ where E is its nonempty finite set of *nodes*, the $\lhd_c \subseteq E \times E$ are disjoint binary relations on E, and $\lambda : E \to \Sigma$ is a *(node-)labeling function*.

Thus, we consider a node $e \in E$ of a graph $G = (E, \{\lhd_c\}_{c \in C}, \lambda)$ over (Σ, C) to be labeled with a letter $a \in \Sigma$ if $\lambda(e) = a$ and we consider a pair $(e, e') \in \bigcup_{c \in C} \lhd_c$ to be labeled with $c \in C$ if $(e, e') \in \lhd_c$. In the sequel, we call $\lhd := \bigcup_{c \in C} \lhd_c$ the *edge relation* or the set of *edges* of G. Moreover, we sometimes write $\leq_c$ for $(\lhd_c)^*$, abbreviate $(\lhd_c)^+$ by $<_c$, set $\leq$ to be the relation $\lhd^*$, and abbreviate $\lhd^+$ by $<$. We call G *connected* if, for any $e, e' \in E$, $(e, e') \in (\lhd \cup \lhd^{-1})^*$. The *cardinality* of G, denoted by $|G|$, is actually meant to be the cardinality $|E|$ of E. Moreover, for a subset Σ' of Σ, we set $|G|_{\Sigma'}$ to be $|\lambda^{-1}(\Sigma')|$. Given $a \in \Sigma$, we then abbreviate $|G|_{\{a\}}$ by $|G|_a$.

Graphs will primarily serve as a convenient representation of partial orders, which, in turn, are a general model for the behavior of a distributed system. Thus, we assume in the sequel a graph $(E, \{\lhd_c\}_{c \in C}, \lambda)$ to generate a partial order, which means that $(E, \lhd^*)$ is supposed to be a partially ordered set. We furthermore require $\lhd$ to be irreflexive. The set of all those *acyclic* graphs is denoted by $\mathbb{DG}(\Sigma, C)$. A useful subclass of $\mathbb{DG}(\Sigma, C)$, denoted by $\mathbb{DG}_H(\Sigma, C)$, is the set of graphs $(E, \{\lhd_c\}_{c \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ such that $\lhd = \lessdot$, i.e., $(E, \lhd)$ is the Hasse diagram of some partially ordered set. Usually, this will be required if the set of edge labelings is a singleton. Throughout the thesis, the nodes of a graph are called *events* executing *actions*, which are given by their node labeling.

Consider the graph shown in Figure 2.1 (a). It is contained in $\mathbb{DG}_H(\{a, b\}, \{1, 2\})$, as its edge relation is a minimal one to generate some partial order. In contrast, the graph from Figure 2.1 (b), though it is contained in $\mathbb{DG}(\{a, b\}, \{1, 2\})$, is not minimal in this sense, as there is an edge from e_1 to e_3 that is already implicitly given by (e_1, e_2) and (e_2, e_3) (even if their common edge labeling, which is 1, is different from the one of (e_1, e_3), which is 2). Thus, Figure 2.1 (b) illustrates a graph that is not contained in $\mathbb{DG}_H(\{a, b\}, \{1, 2\})$. Finally, the structure from Figure 2.1 (c) is not even contained in $\mathbb{DG}(\{a, b\}, \{1, 2\})$, as it is not acyclic.

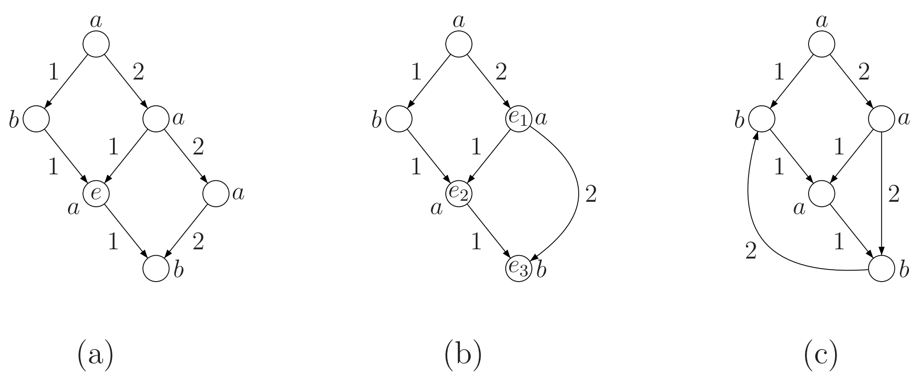


Figure 2.1: Graphs over $(\{a, b\}, \{1, 2\})$

For $G = (E, \{\lhd_c\}_{c \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$, a nonempty subset Σ' of Σ with $\lambda^{-1}(\Sigma') \neq \emptyset$, and $c \in C$, we denote by $G \restriction (\Sigma', \{c\})$ (we just write $G \restriction \Sigma'$ if C is a singleton) the *projection* $(E', \lhd'_c, \lambda') \in \mathbb{DG}(\Sigma', \{c\})$ of G onto Σ' and c where

- $E' = \lambda^{-1}(\Sigma')$,
- $\lhd'_c$ is the union of $\lhd_c \cap (E' \times E')$ and the cover relation of the partial order $(\lhd_c)^* \cap (E' \times E')$, and
- $\lambda' = \lambda_{|E'}$ (i.e., λ' is the restriction of λ to E').

For $e \in E$, let furthermore $G \Downarrow e$ stand for the *downwards closure* of G wrt. e, i.e., for the graph $(E', \{\lhd'_c\}_{c \in C}, \lambda') \in \mathbb{DG}(\Sigma, C)$ where $E' = \{e' \in E \mid e' \lhd^* e\}$, $\lhd'_c = \lhd_c \cap (E' \times E')$, and $\lambda' = \lambda_{|E'}$. If $\{e' \in E \mid e' \lhd e\} \neq \emptyset$, the *strict downwards closure* of G wrt. e, denoted by $G \downarrow e$, is obtained in the same way as the downwards closure, now taking $E' = \{e' \in E \mid e' \lhd^+ e\}$ as a starting point. Again, consider Figure 2.1 on the page before to exemplify projection and downwards closure. The projection $G \upharpoonright (\{a\}, \{1\})$ of the graph $G \in \mathbb{DG}(\{a, b\}, \{1, 2\})$ from Figure 2.1 (a) onto $\{a\}$ and 1 is depicted in Figure 2.2 (a). Note that the projection of a graph from $\mathbb{DG}_H(\Sigma, C)$ is necessarily contained in $\mathbb{DG}_H(\Sigma, C)$, too. The downwards closure of G wrt. e is given by Figure 2.2 (b), its strict downwards closure is depicted aside, in Figure 2.2 (c).

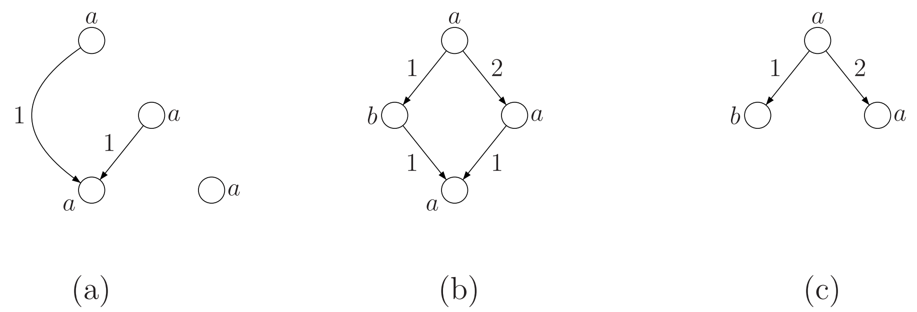


Figure 2.2: The projection and the downwards closure of a graph

Let B be a natural. For $G = (E, \{\lhd_c\}_{c \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$, we say that the degree of G is *bounded* by B if, for any $e \in E$, $|\{e' \in E \mid e \lhd e' \text{ or } e' \lhd e\}| \leq B$. Given $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$, the *degree* of $\mathcal{K}$ is said to be *bounded* by B if, for any $G \in \mathcal{K}$, the degree of G is bounded by B. We say that $\mathcal{K}$ has *bounded degree* if its degree is bounded by some B.

It will be useful to define *extended* graphs, whose nodes are equipped with an additional labeling. Let Q be a nonempty and finite set. A (Q-)*extended graph* over (Σ, C) is a graph $(E, \{\lhd_c\}_{c \in C}, \lambda) \in \mathbb{DG}(\Sigma \times Q, C)$, i.e, λ is a mapping $E \to \Sigma \times Q$. Note that λ can be seen as a pair (λ', ρ) of mappings $E \to \Sigma$ and $E \to Q$, respectively. Given a class $\mathcal{K}$ of graphs over (Σ, C), the corresponding set of Q-extensions is denoted by $\mathcal{K}^Q$.

2.2.2 Monadic Second-Order Logic over Graphs

We recall the notion of monadic second-order (MSO) logic over graphs, i.e., in its most general case, which then carries over to the more specific cases of words, traces, and message sequence charts. Fragments of MSO will give logical characterizations of respective automata models. For a comprehensive overview of monadic second-order logics, see [GTW02].

Throughout the thesis, we fix supplies $\mathrm{Var} = \{x, y, \ldots, x_1, x_2, \ldots\}$ of *individual variables* and $\mathrm{VAR} = \{X, Y, \ldots, X_1, X_2, \ldots\}$ of *set variables*.

Definition 2.2.2 (Monadic Second-Order Logic over Graphs)
Formulas from $\mathrm{MSO}(\Sigma, C)$, the set of *monadic second-order formulas* over the class $\mathbb{DG}(\Sigma, C)$, are built up from the atomic formulas

- $\lambda(x) = a \quad$ (for $a \in \Sigma$),
- $x \lhd_c y \quad$ (for $c \in C$),
- $x \in X$, and
- $x = y$

(where $x, y \in \mathrm{Var}$ and $X \in \mathrm{VAR}$) and, furthermore, allow the boolean connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$, and $\leftrightarrow$ and the quantifiers $\exists$ and $\forall$, which can be applied to either kind of variable and are called individual (first-order) and set (second-order) quantifier, respectively.

Let $x \lhd y$ stand for the $\mathrm{MSO}(\Sigma, C)$-formula $\bigvee_{c \in C} x \lhd_c y$. Moreover, for some $c \in C$, $x \leq_c y$ and $x \leq y$ will abbreviate

$$\forall X \left(x \in X \wedge \forall z \forall z' \left((z \in X \wedge z \lhd_c z') \rightarrow z' \in X\right) \rightarrow y \in X\right)$$

and, respectively,

$$\forall X \left(x \in X \wedge \forall z \forall z' \left((z \in X \wedge z \lhd z') \rightarrow z' \in X\right) \rightarrow y \in X\right),$$

while $x <_c y$ and $x < y$ are shorthands for the formulas $x \leq_c y \wedge \neg(x = y)$ and $x \leq y \wedge \neg(x = y)$, respectively.

Let $G = (E, \{\lhd_c\}_{c \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ be a graph. Given an interpretation function $\mathcal{I}$, which assigns to an individual variable x an event $\mathcal{I}(x) \in E$ and to a set variable X a set of events $\mathcal{I}(X) \subseteq E$, the satisfaction relation $G \models_{\mathcal{I}} \varphi$ for a formula $\varphi \in \mathrm{MSO}(\Sigma, C)$ is given by

- $G \models_{\mathcal{I}} \lambda(x) = a \quad$ if $\lambda(\mathcal{I}(x)) = a$,
- $G \models_{\mathcal{I}} x \lhd_c y \quad$ if $\mathcal{I}(x) \lhd_c \mathcal{I}(y)$,
- $G \models_{\mathcal{I}} x \in X \quad$ if $\mathcal{I}(x) \in \mathcal{I}(X)$, and

– $G \models_{\mathcal{I}} x = y$ if $\mathcal{I}(x) = \mathcal{I}(y)$,

while the remaining operators are defined as usual. If we consider *sentences*, i.e., formulas without free variables, we accordingly replace $\models_{\mathcal{I}}$ with $\models$.

For an $\mathrm{MSO}(\Sigma, C)$-formula φ, the notation $\varphi(x_1, \ldots, x_m, X_1, \ldots, X_n)$ shall indicate that at most $x_1, \ldots, x_m, X_1, \ldots, X_n$ occur free in φ. The fragment of $\mathrm{MSO}(\Sigma, C)$ that does not make use of any set quantifier is the set of *first-order formulas* over $\mathbb{DG}(\Sigma, C)$ and denoted by $\mathrm{FO}(\Sigma, C)$. An $\mathrm{MSO}(\Sigma, C)$-formula is called *existential* if it is of the form $\exists X_1 \ldots \exists X_n \varphi(X_1, \ldots, X_n, \overline{Y})$ where $\overline{Y}$ is a block of second-order variables and $\varphi(X_1, \ldots, X_n, \overline{Y}) \in \mathrm{FO}(\Sigma, C)$. Let $\mathrm{EMSO}(\Sigma, C)$ denote the class of existential $\mathrm{MSO}(\Sigma, C)$-formulas. In general, we would like to distinguish formulas by their quantifier-alternation depth. So $\Sigma_k(\Sigma, C)$ $(k \geq 1)$ shall contain the $\mathrm{MSO}(\Sigma, C)$-formulas of the form $\exists \overline{X_1} \forall \overline{X_2} \ldots \exists / \forall \overline{X_k} \varphi(\overline{X_1}, \ldots, \overline{X_k}, \overline{Y})$ with first-order kernel $\varphi(\overline{X_1}, \ldots, \overline{X_k}, \overline{Y})$ ($\overline{X_i}$ and $\overline{Y}$ are blocks of second-order variables). Note that $\Sigma_1(\Sigma, C)$ and $\mathrm{EMSO}(\Sigma, C)$ coincide. Let us furthermore introduce a variant of $\mathrm{MSO}(\Sigma, C)$: choosing our atomic entities to be

$$\lambda(x) = a \text{ (for } a \in \Sigma) \qquad x \leq y \qquad x \in X \qquad x = y$$

yields respectively the logics $\mathrm{MSO}(\Sigma, C)[\leq]$, $\mathrm{EMSO}(\Sigma, C)[\leq]$, $\Sigma_k(\Sigma, C)[\leq]$, and $\mathrm{FO}(\Sigma, C)[\leq]$. The semantics of $x \leq y$ wrt. a graph $G = (E, \{\lhd_c\}_{c \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ and a corresponding interpretation function $\mathcal{I}$ is determined by $G \models_{\mathcal{I}} x \leq y$ if $\mathcal{I}(x) \lhd^* \mathcal{I}(y)$. Other logics arise in a similar manner. Restricting to an ordering relation $\lessdot$ might lead to the logic $\mathrm{EMSO}(\Sigma, C)[\lessdot]$, for example. Otherwise, it will be clear from the context, which predicates are supported by a logic and which are not.

For set variables $X_1, \ldots, X_n$ $(n \geq 1)$, the first-order formula

$$\mathrm{partition}(X_1, \ldots, X_n) := \left(\forall x \bigvee_{i \in \{1, \ldots, n\}} x \in X_i \right) \wedge \left(\forall x \bigwedge_{1 \leq i < j \leq n} \neg (x \in X_i \wedge x \in X_j) \right)$$

will subsequently formalize that the set of nodes of the graph at hand can be partitioned into sets $X_1, \ldots, X_n$.

Let $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$. For an $\mathrm{MSO}(\Sigma, C)$-sentence φ, the *language of* φ relative to $\mathcal{K}$, denoted by $L_{\mathcal{K}}(\varphi)$, is the set of graphs $G \in \mathcal{K}$ with $G \models \varphi$. However, as a formula $\varphi(X_1, \ldots, X_n) \in \mathrm{MSO}(\Sigma, C)$ (with free variables) can be considered to define a language of graphs whose labelings are enriched by tuples from $\{0, 1\}^n$, we may accordingly denote the corresponding language of φ relative to $\mathcal{K}$ by $L_{\mathcal{K}}(\varphi)$, too, which is then a subset of $\mathcal{K}^{\{0,1\}^n}$. More precisely, an extended graph $G = (E, \{\lhd_c\}_{c \in C}, (\lambda, \rho)) \in \mathcal{K}^{\{0,1\}^n}$ *satisfies* φ if we have $(E, \{\lhd_c\}_{c \in C}, \lambda) \models_{\mathcal{I}_G} \varphi$

where, for any $e \in E$, $e \in \mathcal{I}_G(X_i)$ if $\rho(e)[i] = 1$ (where $\rho(e)[i]$ yields the i-th component of $\rho(e)$).

For $\mathfrak{F} \subseteq \mathrm{MSO}(\Sigma, C)$ and sets $L, \mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$, L is called $\mathfrak{F}_{\mathcal{K}}$*-definable* if $L = L_{\mathcal{K}}(\varphi)$ for some sentence $\varphi \in \mathfrak{F}$. The induced classes of $\mathrm{MSO}(\Sigma, C)_{\mathcal{K}}$-, $\mathrm{EMSO}(\Sigma, C)_{\mathcal{K}}$-, $\Sigma_k(\Sigma, C)_{\mathcal{K}}$-, and $\mathrm{FO}(\Sigma, C)_{\mathcal{K}}$-definable graph languages are denoted by $\mathcal{MSO}(\Sigma, C)_{\mathcal{K}}$, $\mathcal{EMSO}(\Sigma, C)_{\mathcal{K}}$, $\mathcal{L}_{\mathcal{K}}(\Sigma_k(\Sigma, C))$, and $\mathcal{FO}(\Sigma, C)_{\mathcal{K}}$, respectively. Accordingly, wrt. the alternative predicate symbol $\leq$, we obtain further classes of graph languages, namely $\mathcal{MSO}(\Sigma, C)[\leq]_{\mathcal{K}}$, $\mathcal{EMSO}(\Sigma, C)[\leq]_{\mathcal{K}}$, $\mathcal{L}_{\mathcal{K}}(\Sigma_k(\Sigma, C)[\leq])$, and $\mathcal{FO}(\Sigma, C)[\leq]_{\mathcal{K}}$.

For $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$, we say that the *monadic quantifier-alternation hierarchy* over $\mathcal{K}$ is infinite if the sets $\mathcal{L}_{\mathcal{K}}(\Sigma_k(\Sigma, C))$, $k = 1, 2, \ldots$, form an infinite strict hierarchy. Recall that, in general, the classes of $\Sigma_k(\Sigma, C)_{\mathbb{DG}(\Sigma,C)}$-definable languages form an infinite hierarchy [MT97, MST02]. In other words, the more alternation of second-order variables is allowed, the more expressive formulas become.

It may be the case that the set of node labelings or the set of edge labelings is a singleton so that we do not need to explicitly refer to Σ and C, respectively. In that case, we speak of graphs over $(\Sigma, -)$ or over $(-, C)$ and respectively write, for example, $\mathbb{DG}(\Sigma, -)$ and $\mathcal{FO}(-, C)[\leq]_{\mathcal{K}}$. Moreover, if the labeling alphabets are clear from the context, we often omit the reference to Σ and C completely and write, for instance, $\mathbb{DG}$, EMSO, $\mathcal{MSO}_{\mathcal{K}}$, or $\mathcal{FO}[\leq]_{\mathcal{K}}$.

Besides $\mathrm{MSO}(\Sigma, C)[\leq]$ and corresponding sublogics, consider $\mathrm{MSO}_0(\Sigma, C)$, another slightly modified logic, which, in contrast to $\mathrm{MSO}(\Sigma, C)[\leq]$, has in general the same expressive power as $\mathrm{MSO}(\Sigma, C)$. Its atomic entities are

$$\lambda(X) \subseteq \{a\} \text{ (for } a \in \Sigma) \qquad X \lhd_c Y \qquad X \subseteq Y \qquad \mathit{Sing}(X)$$

where $X, Y \in \mathrm{VAR}$. Moreover, only second-order quantifiers are allowed. The meaning of an $\mathrm{MSO}_0(\Sigma, C)$-formula is the expected one, i.e., given a graph $G = (E, \{\lhd_c\}_{c \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ and an interpretation $\mathcal{I}$,

- $G \models_{\mathcal{I}} \lambda(X) \subseteq \{a\}$ if, for any $e \in \mathcal{I}(X)$, $\lambda(e) = a$,
- $G \models_{\mathcal{I}} X \lhd_c Y$ if $\mathcal{I}(X)$ and $\mathcal{I}(Y)$ are singletons $\{e\}$ and, respectively, $\{e'\}$ such that $e \lhd_c e'$,
- $G \models_{\mathcal{I}} X \subseteq Y$ if $\mathcal{I}(X) \subseteq \mathcal{I}(Y)$, and
- $G \models_{\mathcal{I}} \mathit{Sing}(X)$ if $\mathcal{I}(X)$ is a singleton.

Following our convention, let, given a set $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ of graphs, $\mathcal{MSO}_0(\Sigma, C)_{\mathcal{K}}$ denote the class of $\mathrm{MSO}_0(\Sigma, C)_{\mathcal{K}}$-definable graph languages. It is easy to construct from an $\mathrm{MSO}(\Sigma, C)$- an equivalent $\mathrm{MSO}_0(\Sigma, C)$-sentence and vice versa, which leads us to the following lemma.

Lemma 2.2.3 For any $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$, we have

$$\mathcal{MSO}(\Sigma, C)_{\mathcal{K}} = \mathcal{MSO}_0(\Sigma, C)_{\mathcal{K}}.$$

Let us discuss further relations between the proposed language classes.

Lemma 2.2.4 Let L and $\mathcal{K}$ be subsets of $\mathbb{DG}$. If L is $\mathrm{MSO}_{\mathcal{K}}$-definable, then it is $(\Sigma_k)_{\mathcal{K}}$-definable for some $k \geq 1$.

Proof From a given MSO-sentence φ, we first build an MSO_0-sentence φ' in prenex normal form that is equivalent to φ wrt. graphs from $\mathcal{K}$. In particular, each first-order quantifier has been replaced with a second-order one, while the resulting second-order variables X have been relativized by the (first-order definable) predicate $\mathit{Sing}(X)$. First-order definability of the atomic predicates occurring in φ' then leads to some Σ_k-formula for suitable k. $\square$

Lemma 2.2.5 For any class $\mathcal{K} \subseteq \mathbb{DG}$, we have

$$\mathcal{MSO}[\leq]_{\mathcal{K}} \ \subseteq \ \mathcal{MSO}_{\mathcal{K}}.$$

Proof In an MSO[$\leq$]-formula, replace any occurrences of $x \leq y$ with

$$\forall X\,(x \in X \wedge \forall z \forall z'\,((z \in X \wedge z \lhd z') \rightarrow z' \in X) \rightarrow y \in X)$$

to obtain an MSO-formula. $\square$

However, the inverse does not hold in general, i.e., the collection of $\lhd_c$ is not necessarily expressible in terms of $\leq$. As we will see, the above does neither hold for the existential nor the first-order fragment of MSO.

Trivially, the sets of first-order, existential monadic second-order, and monadic second-order formulas form a hierarchy:

Lemma 2.2.6 For any class $\mathcal{K} \subseteq \mathbb{DG}$, we have

(a) $\mathcal{FO}_{\mathcal{K}} \ \subseteq \ \mathcal{EMSO}_{\mathcal{K}} \ \subseteq \ \mathcal{MSO}_{\mathcal{K}}$ and

(b) $\mathcal{FO}[\leq]_{\mathcal{K}} \ \subseteq \ \mathcal{EMSO}[\leq]_{\mathcal{K}} \ \subseteq \ \mathcal{MSO}[\leq]_{\mathcal{K}}$.

2.2.3 Hanf's Theorem

Besides formulas, graphs themselves may provide a framework to specify graph properties. For instance, we might be interested in the set of those graphs in which a given pattern occurs at least, say, $n \in \mathbb{N}$ times. A pattern H hereby specifies the local neighborhood around a distinguished center γ where the size of the neighborhood is constituted by a natural $R \in \mathbb{N}$, the *radius* of H, which restricts the distance of any node of H to γ.

Let us make this idea more precise and let R be a natural. Given a graph $G = (E, \{\lhd_c\}_{c\in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ and nodes $e, e' \in E$, the *distance* $d_G(e', e)$ from e' to e in G is ∞ if it holds $(e, e') \notin (\lhd \cup \lhd^{-1})^*$ and, otherwise, the minimal natural number k such that there is a sequence of elements $e_0, \ldots, e_k \in E$ with $e_0 = e$, $e_k = e'$, and $e_i \lhd e_{i+1}$ or $e_{i+1} \lhd e_i$ for each $i \in \{0, \ldots, k-1\}$. Sometimes, if it is clear from the context, we omit the subscript G just writing $d(e', e)$. An *R-sphere* over (Σ, C) is a graph $H = (E, \{\lhd_c\}_{c\in C}, \lambda, \gamma)$ over (Σ, C) together with a designated *sphere center* $\gamma \in E$ such that, for any $e \in E$, $d_H(e, \gamma) \leq R$ (in slight abuse of notation, the distance from one node to another can be given wrt. a sphere as well). For a graph $G = (E, \{\lhd_c\}_{c\in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ and $e \in E$, let the *R-sphere of G around e*, denoted by $R\text{-Sph}(G, e)$, be given by $(E', \{\lhd'_c\}_{c\in C}, \lambda', e)$ where $E' = \{e' \in E \mid d_G(e', e) \leq R\}$, $\lhd'_c = \lhd_c \cap (E' \times E')$ for each $c \in C$, and λ' is the restriction of λ to E'. A 2-sphere over $(\{a, b\}, \{1, 2\})$ is shown in part (a) of Figure 2.3 where the sphere center is depicted as a rectangle. It precisely deals with the 2-sphere of the graph aside (Figure 2.3 (b)) around e.

Essentially, Hanf's Theorem states that any first-order sentence can be rephrased as a boolean combination of conditions "R-sphere H occurs at least $n \in \mathbb{N}$ times". Such a requirement is said to be satisfied by $G = (E, \{\lhd_c\}_{c\in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ if, in the obvious manner, it is satisfied by the mapping that assigns to each R-sphere H the natural $|\{e \in E \mid R\text{-Sph}(G, e) \cong H\}|$, i.e, the number of occurrences of H in G. (Hereby and henceforth, $\cong$ stands for the isomorphism relation.) Relative to a set of graphs $\mathcal{K}$, this gives rise to the class of *locally threshold testable* graph languages, which is denoted by $\mathcal{LTT}(\Sigma, C)_{\mathcal{K}}$.

Theorem 2.2.7 ([Han65]) For any class $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ of bounded degree and any $L \subseteq \mathcal{K}$, $L \in \mathcal{FO}(\Sigma, C)_{\mathcal{K}}$ implies $L \in \mathcal{LTT}(\Sigma, C)_{\mathcal{K}}$.

The next step towards a logically founded automata theory over graphs is to establish a connection between EMSO-definable and locally threshold testable languages. So let us introduce some further notions and let $n \geq 1$ be a positive natural. Given an extended graph $G = (E, \{\lhd_c\}_{c\in C}, \lambda) \in \mathbb{DG}(\Sigma \times \{0,1\}^n, C)$ over (Σ, C), the *projection* of G is defined to be $h(G) := (E, \{\lhd_c\}_{c\in C}, \lambda') \in \mathbb{DG}(\Sigma, C)$ where, for any $e \in E$, we have $\lambda'(e) = a$ if $\lambda(e) = (a, (b_1, \ldots, b_n))$ for some $b_1, \ldots, b_n \in \{0, 1\}$. (Note that this differs from the definition of a projection we have given above.) The projection function h is canonically extended towards graph languages $L \subseteq \mathbb{DG}(\Sigma \times \{0,1\}^n, C)$, i.e., $h(L) := \{h(G) \mid G \in L\}$. Recall that a formula $\varphi(X_1, \ldots, X_n) \in \text{MSO}(\Sigma, C)$ can be interpreted wrt. G by inferring

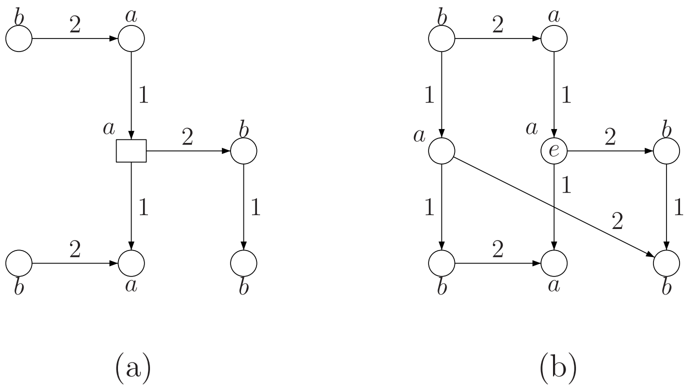


Figure 2.3: A 2-sphere over $(\{a, b\}, \{1, 2\})$

from the additional labelings an interpretation function $\mathcal{I}_G$, which assigns to variable X_i all those nodes whose labeling in the i-th component equals 1 so that we may write $G \models \varphi(X_1, \ldots, X_n)$ if $h(G) \models_{\mathcal{I}_G} \varphi(X_1, \ldots, X_n)$.

The following proposition basically states that a language is EMSO-definable iff it is the projection of some locally threshold testable set.

Proposition 2.2.8 ([Tho96]) For any class $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ of bounded degree, we have $\mathcal{EMSO}(\Sigma, C)_{\mathcal{K}} = \bigcup_{n \geq 1}\{h(L) \mid L \in \mathcal{LTT}(\Sigma \times \{0,1\}^n, C)_{\mathcal{K}^{\{0,1\}^n}}\}$.

Proof Recall that a graph language $L \subseteq \mathbb{DG}(\Sigma, C)$ is $\text{EMSO}_{\mathcal{K}}$-definable iff it is the language of some EMSO(Σ, C)-sentence $\exists X_1 \ldots \exists X_n \varphi(X_1, \ldots, X_n)$ (where, without loss of generality, we can assume that $n \geq 1$) with first-order kernel φ. As φ can be seen as a sentence from FO$(\Sigma \times \{0,1\}^n, C)$ (replace any $x \in X_i$ with $\bigvee_{b_i=1} \lambda(x) = (a, (b_1, \ldots, b_n))$, $L_{\mathcal{K}^{\{0,1\}^n}}(\varphi) \in \mathcal{FO}(\Sigma \times \{0,1\}^n, C)_{\mathcal{K}^{\{0,1\}^n}}$ is, according to Theorem 2.2.7, a locally threshold testable set, whose projection $h(L_{\mathcal{K}^{\{0,1\}^n}}(\varphi))$ coincides with L. $\square$

2.2.4 Graph Acceptors

Graph acceptors [Tho91, Tho97a] are a generalization of finite automata to graphs. They are known to be expressively equivalent to EMSO logic wrt. graphs of bounded degree. A graph acceptor works on a graph as follows: it first assigns to each node one of its control states and then checks if the local neighborhood of each node (incorporating the state assignment) corresponds to a pattern from a finite supply of spheres.

Definition 2.2.9 (Graph Acceptor [Tho91, Tho97a])
A *graph acceptor* over (Σ, C) is a structure $\mathcal{B} = (Q, R, \mathfrak{S}, Occ)$ where

- Q is its nonempty finite set of *states*,
- $R \in \mathbb{N}$ is the *radius*,
- $\mathfrak{S}$ is a finite set of R-spheres over $(\Sigma \times Q, C)$, and
- Occ is a boolean combination of *conditions* of the form "sphere $H \in \mathfrak{S}$ occurs at least n times" where $n \in \mathbb{N}$.

A *run* of $\mathcal{B}$ on a graph $G = (E, \{\lhd_c\}_{c\in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ is a mapping $\rho : E \to Q$ such that, for each $e \in E$, the R-sphere of $(E, \{\lhd_c\}_{c\in C}, (\lambda, \rho))$ around e is isomorphic to some $H \in \mathfrak{S}$. We call ρ *accepting* if the tiling of G with spheres from $\mathfrak{S}$, which is uniquely determined by ρ, satisfies the constraints imposed by Occ. (In the tiling induced by ρ, the sphere $H \in \mathfrak{S}$ occurs $|\{e \in E \mid H \cong R\text{-Sph}((E, \{\lhd_c\}_{c\in C}, (\lambda, \rho)), e)\}|$ times.) The language of $\mathcal{B}$ relative to a class $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$, denoted by $L_{\mathcal{K}}(\mathcal{B})$, is the set of graphs $G \in \mathcal{K}$ on which there is an accepting run of $\mathcal{B}$. Moreover, we denote by $\mathcal{GA}(\Sigma, C)_{\mathcal{K}}$ ($\mathcal{GA}_{\mathcal{K}}$ if Σ and C are clear from the context) the class $\{L \subseteq \mathcal{K} \mid L = L_{\mathcal{K}}(\mathcal{B})$ for some graph acceptor $\mathcal{B}$ over $(\Sigma, C)\}$. An interesting class of graph languages distinguishes those sets that are recognized by some graph acceptor that employs only k-spheres for some $k \in \mathbb{N}$. Accordingly, we denote by k-$\mathcal{GA}(\Sigma, C)_{\mathcal{K}}$ or k-$\mathcal{GA}_{\mathcal{K}}$ the class $\{L \subseteq \mathcal{K} \mid L = L_{\mathcal{K}}(\mathcal{B})$ for some graph acceptor $\mathcal{B} = (Q, R, \mathfrak{S}, Occ)$ over (Σ, C) with $R = k\}$.

The following is quite obvious:

Lemma 2.2.10 For any $k \in \mathbb{N}_{\geq 1}$ and $\mathcal{K} \subseteq \mathbb{DG}$,

$$k\text{-}\mathcal{GA}_{\mathcal{K}} \subseteq (k+1)\text{-}\mathcal{GA}_{\mathcal{K}}.$$

Conversely, however, the radius of some graph acceptor cannot be reduced arbitrarily.

Lemma 2.2.11

$$1\text{-}\mathcal{GA}_{\mathbb{DG}} \subsetneqq \mathcal{GA}_{\mathbb{DG}}.$$

The proof is deferred to Section 2.5.

There are graph languages recognized by graph acceptors that can even do without any occurrence constraint, which actually means that the occurrence constraint is set to be true a priori. We denote the arising classes by $\mathcal{GA}(\Sigma, C)^{-}_{\mathcal{K}}$ ($\mathcal{GA}^{-}_{\mathcal{K}}$) and k-$\mathcal{GA}(\Sigma, C)^{-}_{\mathcal{K}}$ (k-$\mathcal{GA}^{-}_{\mathcal{K}}$).

Lemma 2.2.12 In general, $1\text{-}\mathcal{GA}_{\mathcal{K}} \setminus \mathcal{GA}^{-}_{\mathcal{K}}$ is not empty.

Proof Consider graphs G_1 and G_2 over $(\{a,b\},-)$ where G_1 and G_2 each consist of one event, which is labeled with a and b, respectively. Then, $\{G_1, G_2\}$ is contained in $1\text{-}\mathcal{GA}_{\mathbb{DG}(\{a,b\},-)} \setminus \mathcal{GA}^{-}_{\mathbb{DG}(\{a,b\},-)}$. On the one hand, any graph acceptor without occurrence constraint recognizing both G_1 and G_2 also admits an accepting run on the union of G_1 and G_2 (which is obtained in the obvious manner). On the other hand, $\{G_1, G_2\}$ is the language of some graph acceptor with radius 1 where an occurrence constraint ensures that the union of G_1 and G_2 is excluded from the recognized language. $\square$

Note that, considering a graph acceptor relative to the class $\mathbb{DG}$ of all graphs, its spheres themselves are contained in $\mathbb{DG}$. It might be worth noting that such a coincidence does not necessarily hold for arbitrary classes of graphs, i.e., applying graph acceptors to a subclass $\mathcal{K}$ of $\mathbb{DG}$, their spheres might still require a more general structure than $\mathcal{K}$ admits. But obviously, it always suffices to restrict to those spheres that can be *embedded* into some graph from $\mathcal{K}$. Those considerations will play a role when we address the issue of graph acceptors over message sequence charts.

Let us now compare EMSO logic to the formalism of graph acceptors.

Theorem 2.2.13 ([Tho96, Tho97a]) For any class $\mathcal{K} \subseteq \mathbb{DG}$ of bounded degree, it holds

$$\mathcal{EMSO}_{\mathcal{K}} \; = \; \mathcal{GA}_{\mathcal{K}}.$$

Proof The equivalence directly follows from Proposition 2.2.8. In particular, the number of states of a graph acceptor simulating a given existential sentence $\exists X_1 \ldots \exists X_n \varphi(X_1, \ldots, X_n)$ with first-order kernel φ depends on the number n of set quantifiers. $\square$

2.3 Words

Words can be represented in many different ways. For example, a word can be seen as a string, i.e., a sequence of symbols. Such a sequence gives rise to a totally ordered set, a special case of a partially ordered one, which, as we have seen, has a graph-theoretical counterpart.

Let Σ be an alphabet. A *word* over Σ is an (in our case) nonempty structure $w = (\{1, \ldots, n\}, \lessdot, \lambda)$ where $1, \ldots, n$ are the *letter positions* of w, $\lessdot$ is the *successor relation* on $\{1, \ldots, n\}$, which contains the pairs $(i, i+1)$ with $i \in \{1, \ldots, n-1\}$, and λ is a mapping $\{1, \ldots, n\} \to \Sigma$. The set of words over Σ is denoted by $\mathbb{W}(\Sigma)$

or simply by $\mathbb{W}$ if Σ is clear from the context. Note that a word over Σ is just a graph over Σ and a singleton, which is, to some extent, extraneous. Thus, $\mathbb{W}(\Sigma)$ is simply $\mathbb{DG}(\Sigma, -)$ restricted to graphs $(E, \lhd, \lambda) \in \mathbb{DG}_H(\Sigma, -)$ such that $\lhd^*$ forms a total order. Recall that we do not distinguish isomorphic structures. We can therefore identify a word $(\{1, \ldots, n\}, \lhd, \lambda) \in \mathbb{W}$ with the sequence $a_1 \ldots a_n \in \Sigma^*$ such that, for each $i \in \{1, \ldots, n\}$, $a_i = \lambda(i)$. Given a word $(\{1, \ldots, n\}, \lhd, \lambda) \in \mathbb{W}$, we denote by *first*$(w)$ and *last*(w) the events 1 and n, respectively.

It will be useful to allow a word to be *empty*. The empty word, denoted by ε in the following, can be considered to be the structure $(\emptyset, \emptyset, \emptyset)$, i.e., we deal with the structure with an empty set of events. Though, in subsequent definitions, the empty word is not addressed explicitly as a special case of a word, it will be clear how to extend definitions towards ε. Moreover, ε appears as the unit word in the free monoid $\mathbb{W}$. Recognizability and rationality coincide in finitely-generated free monoids, which is the commonly known Kleene's Theorem.

Theorem 2.3.1 ([Kle56])

$$\mathcal{REC}_{\mathbb{W}} = \mathcal{RAT}_{\mathbb{W}}$$

Note that a recognizable word language is also called *regular*. As $\mathbb{W}(\Sigma) \subseteq \mathbb{DG}(\Sigma, -)$, the monadic second-order formulas that can be applied to words over Σ, are those from $\mathrm{MSO}(\Sigma, -)$. Recall that the corresponding atomic entities are

$$\lambda(x) = a \text{ (for } a \in \Sigma) \qquad x \lhd y \qquad x \in X \qquad x = y$$

(where $x, y \in \mathrm{Var}$ and $X \in \mathrm{VAR}$). The definition of their semantics arises from the general case of graphs.

We now recall a well-known automata model, which is tailored to words.

Definition 2.3.2 (Finite Automaton)
A *finite automaton* over Σ is a structure (S, Δ, s^{in}, F) where

- S is its nonempty finite set of *states*,
- $\Delta \subseteq S \times \Sigma \times S$ is the set of *transitions*,
- $s^{in} \in S$ is the *initial state*, and
- $F \subseteq S$ is the set of final states.

Let $\mathcal{A} = (S, \Delta, s^{in}, F)$ be a finite automaton over Σ. A *run* of $\mathcal{A}$ on a word $w = (\{1, \ldots, n\}, \lhd, \lambda) \in \mathbb{W}$ is a mapping $\rho : \{0, 1, \ldots, n\} \to S$ such that $\rho(0) = s^{in}$

and, for each $i \in \{1, \ldots, n\}$, $(\rho(i-1), \lambda(i), \rho(i)) \in \Delta$. We call ρ *accepting* if $\rho(n) \in F$. The *language* of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set $\{w \in \mathbb{W} \mid$ there is an accepting run of $\mathcal{A}$ on $w\}$. Note that ε is defined to be contained in $L(\mathcal{A})$ if (and only if) $s^{in} \in F$. By $\mathcal{FA}(\Sigma)$ ($\mathcal{FA}$ if Σ is clear from the context), we denote the class of word languages that are recognized by some finite automaton over Σ. Obviously, a finite automaton gives rise to a $\mathbb{W}$-automaton and vice versa. Moreover, finite automata can be characterized in terms of MSO.

Theorem 2.3.3 ([Büc60, Elg61])

$$\mathcal{MSO}_{\mathbb{W}} = \mathcal{FA} = \mathcal{REC}_{\mathbb{W}}$$

Proof It remains to show the first equation. Let us first address the direction from right to left. So suppose $\mathcal{A} = (S, \Delta, s^{in}, F)$ to be a finite automaton over an alphabet Σ, say with state set $S = \{q_0, \ldots, q_k\}$ where $s^{in} = q_0$. Then, for any word $w = (\{1, \ldots, n\}, \lessdot, \lambda) \in \mathbb{W}(\Sigma)$, we have $w \in L(\mathcal{A})$ iff

$$\begin{aligned} w \models\ & \exists X_0 \ldots \exists X_k \\ & \Big[\quad \mathrm{partition}(X_0, \ldots, X_k) \\ & \wedge\ \forall x(\mathit{last}(x) \rightarrow \textstyle\bigvee_{q_i \in F} x \in X_i) \\ & \wedge\ \forall x \forall y \Big(x \lessdot y \rightarrow \textstyle\bigvee_{(q_i, a, q_j) \in \Delta} (x \in X_i \wedge \lambda(y) = a \wedge y \in X_j) \Big) \\ & \wedge\ \forall x \Big(\mathit{first}(x) \rightarrow \textstyle\bigvee_{(q_0, a, q_i) \in \Delta} (\lambda(x) = a \wedge x \in X_i) \Big) \quad \Big] \end{aligned}$$

Hereby, the predicates $\mathit{first}(x)$ and $\mathit{last}(x)$ are used to abbreviate $\neg\exists y(y < x)$ and $\neg\exists y(x < y)$, i.e., to access the first and the last position of a word, respectively. Note that, if the empty word is not recognized by $\mathcal{A}$, one hast to add a clause $\exists x(x = x)$, as, otherwise, ε will be included in the language of the above formula.

So let us show the converse, i.e., construct from an arbitrary MSO$(\Sigma, -)$-sentence ψ a finite automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L_{\mathbb{W}(\Sigma)}(\psi)$. According to Lemma 2.2.3, it is sufficient to give an inductive translation of an MSO$_0(\Sigma, -)$-formula of the form $\varphi(Y_1, \ldots, Y_n) = (\exists/\neg\exists)X_k \ldots (\exists/\neg\exists)X_1 \varphi'(Y_1, \ldots, Y_n, X_k, \ldots, X_1)$ with quantifier-free φ', which (if $n \geq 1$) defines a word language over $\Sigma \times \{0,1\}^n$ in the obvious manner, into a corresponding finite automaton $\mathcal{A}$ over $\Sigma \times \{0,1\}^n$. For atomic formulas, the translation is straightforward. For example, the finite automaton of a subformula $X_i \subseteq X_j$ just has to check if, in each letter to read, the component that belongs to X_i is 0 or the component of X_j is 1. In the induction step, we can restrict to negation, disjunction, and existential quantification. While the former two refer to the automata theoretic constructions of complementation and union, respectively, the latter results in a projection of the automaton at hand. So suppose $\mathcal{A}$ to be the finite automaton for

$\varphi''(Y_1, \ldots, Y_n, X_k, \ldots, X_{i+1}) = (\exists/\neg\exists)X_i \ldots (\exists/\neg\exists)X_1\varphi'(Y_1, \ldots, Y_n, X_k, \ldots, X_1)$ with $0 \leq i < k$. If we precede φ'' with $\exists X_{i+1}$, then the desired automaton over $\Sigma \times \{0,1\}^{n+k-(i+1)}$ basically simulates $\mathcal{A}$ but guesses (rather than reads) the component of X_{i+1} in a letter. In other words, each letter is projected onto the remaining components. □

The above proof allows to effectively construct from any given MSO-sentence an equivalent EMSO-sentence.

Corollary 2.3.4

$$\mathcal{EMSO}_{\mathbb{W}} = \mathcal{MSO}_{\mathbb{W}}$$

In general, first-order logic is not as expressive as monadic second-order logic. Moreover, the set of $\text{FO}_{\mathbb{W}}$-definable word languages is strictly contained in the class of $\text{FO}[\leq]_{\mathbb{W}}$-definable languages.

Proposition 2.3.5

$$\mathcal{FO}_{\mathbb{W}} \subsetneq \mathcal{FO}[\leq]_{\mathbb{W}} \subsetneq \mathcal{EMSO}_{\mathbb{W}} = \mathcal{MSO}_{\mathbb{W}}$$

Proof While the first strict inclusion is witnessed by the language of the rational expression $a^* \cdot b \cdot a^* \cdot c \cdot a^*$, which is $\text{FO}[\leq]_{\mathbb{W}}$- but not $\text{FO}_{\mathbb{W}}$-definable, the set of words of even length turns out to be $\text{EMSO}_{\mathbb{W}}$- but not $\text{FO}[\leq]_{\mathbb{W}}$-definable. See [Tho97b] for further details. □

An important concept of partially ordered sets is their characterization in terms of *linear extensions* or *linearizations*, which establishes a relationship between posets and words. So let $G = (E, \{\lhd_c\}_{c\in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ be a graph. A word $(E', \lhd', \lambda') \in \mathbb{W}(\Sigma)$ is called a *linearization* of G if $E' = E$, $\lhd'$ is the covering relation of some total order containing $\lhd^*$, and $\lambda' = \lambda$. The set of linearizations of G is denoted by $Lin(G)$. This notion is extended to sets L of graphs according to $Lin(L) := \bigcup_{G\in L} Lin(G)$. For example, *abaa* and *aaba* are the only linearizations of the graph depicted in part (b) of Figure 2.2 on page 14. In turn, one cannot uniquely infer a graph from a given linearization in general, because a linear extension abstracts away some edges and edge labelings. However, in most relevant cases (among them Mazurkiewicz traces and message sequence charts), it is possible to reconstruct a graph from a given linearization, as, for those classes of graphs, the edge relation and its label partitioning is uniquely determined by the rest of the structure.

Let us conclude this section with a comparison of the automata models for words we have seen so far.

Corollary 2.3.6

$$1\text{-}\mathcal{GA}_{\mathbb{W}} \;=\; \mathcal{GA}_{\mathbb{W}} \;=\; \mathcal{FA}$$

Proof The second equality directly follows from Theorem 2.2.13, Theorem 2.3.3, and Corollary 2.3.4. To prove the first equality, we easily verify that the transitions of a finite automaton can be simulated using 1-spheres. In particular, initial/final transitions give rise to 1-spheres whose sphere center has no predecessor/successor, respectively. □

Taking into consideration that, over words, a graph acceptor can count the number of employed spheres up to a certain threshold just by means of control states, we get that graph acceptors can do without occurrence constraints. In general, however, this applies at most to classes of connected graphs (cf. Lemma 2.2.12).

Lemma 2.3.7 ([Tho96])

$$\mathcal{GA}^{-}_{\mathbb{W}} \;=\; \mathcal{GA}_{\mathbb{W}}$$

2.4 Mazurkiewicz Traces

While words abstract from independence and dependence of actions or might be used to model the behavior of purely sequential systems, *Mazurkiewicz traces* preserve some partial-order properties of a distributed system. Given an alphabet Σ of *actions*, they are based on a reflexive and *symmetric* relation $D \subseteq \Sigma \times \Sigma$ (in particular, for any $a, b \in \Sigma$, $(a, b) \in D$ implies $(b, a) \in D$), which is called a *dependence relation* over Σ. The pair (Σ, D) is then called a *dependence alphabet*.

Let $K \geq 1$ be a natural in the following and let $[K]$ denote the set $\{1, \ldots, K\}$ of *agents*. Given (not necessarily disjoint) alphabets $\Sigma_1, \ldots, \Sigma_K$, we call the tuple $(\Sigma_1, \ldots, \Sigma_K)$ a *distributed alphabet*. Elements from Σ_i are understood to be actions that are performed by agent i. Let in the following $\tilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_K)$ be a distributed alphabet and, for $a \in \bigcup_{i \in [K]} \Sigma_i$, $loc(a) := \{i \in [K] \mid a \in \Sigma_i\}$ denote the set of agents that are involved in the action a. A distributed alphabet $\tilde{\Sigma}$ determines the canonical dependence relation $\mathcal{D}_{\tilde{\Sigma}} = (\Sigma, D)$ where $\Sigma = \bigcup_{i \in [K]} \Sigma_i$ and $D = \{(a, b) \in \Sigma \times \Sigma \mid loc(a) \cap loc(b) \neq \emptyset\}$. Thus, actions a and b are understood to be *dependent* if they can both be performed by one and the same sequential agent. Trivially, an action $a \in \Sigma$ depends on itself.

For the rest of this section, we fix a natural $K \geq 1$ and a distributed alphabet $\tilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_K)$ with associated dependence relation $\mathcal{D}_{\tilde{\Sigma}} = (\Sigma, D)$, i.e., in particular, Σ will denote $\bigcup_{i \in [K]} \Sigma_i$.

Definition 2.4.1 (Mazurkiewicz Trace [DR95])
A *(Mazurkiewicz) trace* over $\tilde{\Sigma}$ is a graph $(E, \lhd, \lambda) \in \mathbb{DG}_H(\Sigma, -)$ such that, for any $e, e' \in E$,

- $e \lhd e'$ implies $(\lambda(e), \lambda(e')) \in D$ and
- $(\lambda(e), \lambda(e')) \in D$ implies $e \lhd^* e'$ or $e' \lhd^* e$.

Recall that, for a trace $(E, \lhd, \lambda)$ over $\tilde{\Sigma}$, $\lhd$ and $\lessdot$ coincide.

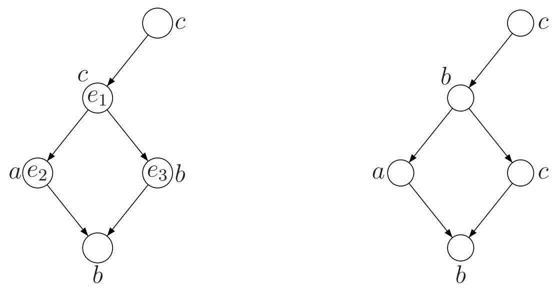
The set of traces over $\tilde{\Sigma}$ is denoted by $\mathbb{TR}(\tilde{\Sigma})$. As usual, we often write $\mathbb{TR}$ if $\tilde{\Sigma}$ can be learned from the context. Remarkably, if $D = \Sigma \times \Sigma$, any trace from $\mathbb{TR}(\tilde{\Sigma})$ constitutes a totally ordered set so that $\mathbb{TR}(\tilde{\Sigma}) \subseteq \mathbb{W}(\Sigma)$.

Accordingly, the structure from part (a) of Figure 2.4 on the next page is not a trace over $(\{a, b\}, \{b, c\})$ (strictly speaking, there is no distributed alphabet at all that makes it a trace). It is contrary to the requirement that events that immediately follow each other carry dependent actions. Though e_2 is an immediate successor of e_1, it executes an action a that is independent of c, which is executed by e_1. This contradicts the intuition that the order of execution only matters if actions clash in using the same resources. For this reason, e_2 and e_3 should actually be ordered. Part (b) of Figure 2.4 on the following page presents a trace over $(\{a, b\}, \{b, c\})$, as its associated partial order fulfills both ordering requirements. Note that, however, two events $e, e' \in E$ of a trace $(E, \lhd, \lambda) \in \mathbb{TR}$ might be ordered wrt. $\lhd^*$ though their labels $\lambda(e)$ and $\lambda(e')$ are independent, per se. This is the case if, for example, there is a third event e'' arranged in order between e and e' whose label depends on both $\lambda(e)$ and $\lambda(e')$.

Note that, usually, Mazurkiewicz traces are defined as posets $(E, \leq, \lambda)$ [DR95]. But to treat all the structures relevant to this thesis in a common framework, a trace is given by its graphical representation. However, from a trace T, we can uniquely derive the corresponding poset T' and, vice versa, T' gives rise to a unique graph, which is identical to T.

2.4.1 Trace Languages

Given a trace $T = (E, \lhd, \lambda) \in \mathbb{TR}$ and $i \in [K]$, let E_i denote $\lambda^{-1}(\Sigma_i)$ and let $T \restriction i$ stand for the empty word if $E_i = \emptyset$ and for $T \restriction \Sigma_i$, otherwise. Note that $T \restriction i \in \mathbb{W}(\Sigma_i)$. For traces $T = (E, \lhd, \lambda)$ and $T' = (E', \lhd', \lambda')$ over $\tilde{\Sigma}$, let $T \cdot T'$ denote the *concatenation* $(E'', \lhd'', \lambda'') \in \mathbb{TR}$ of T and T' where $E'' = E \uplus E'$, $\lambda'' = \lambda \cup \lambda'$, and $\lhd''$ is the cover relation of $(\lhd \cup \lhd' \cup \{(e, e') \in E \times E' \mid (\lambda(e), \lambda(e')) \in D\})^*$. It is an easy task to show that trace concatenation is associative. We artificially add a unit $\mathbf{1}_{\mathbb{TR}}$ to $\mathbb{TR}$, which can be seen as the empty trace. Then, $(\mathbb{TR}, \cdot, \mathbf{1}_{\mathbb{TR}})$ is



(a) not a trace (b) a trace

Figure 2.4: A trace over $(\{a, b\}, \{b, c\})$

a monoid, called the *trace monoid* of (Σ, D), which is mostly identified with $\mathbb{TR}$. Note that, in most definitions, we assume a trace to be nonempty. However, it will be clear how to incorporate $\mathbf{1}_{\mathbb{TR}}$, too, which we do silently.

We have already implicitly defined the classes $\mathcal{MSO}(\Sigma, -)_{\mathbb{TR}(\tilde{\Sigma})}$, $\mathcal{RAT}_{\mathbb{TR}(\tilde{\Sigma})}$, and $\mathcal{REC}_{\mathbb{TR}(\tilde{\Sigma})}$ of $\mathrm{MSO}(\Sigma, -)_{\mathbb{TR}(\tilde{\Sigma})}$-definable, rational, and recognizable trace languages, respectively. Recall that, as $\mathbb{TR}(\tilde{\Sigma}) \subseteq \mathbb{DG}(\Sigma, -)$, the monadic second-order formulas tailored to traces over $\tilde{\Sigma}$ are built from the atomic entities

$$\lambda(x) = a \text{ (for } a \in \Sigma) \qquad x \lhd y \qquad x \in X \qquad x = y$$

(where $x, y \in \mathrm{Var}$ and $X \in \mathrm{VAR}$). For their semantics, see the semantics of $\mathrm{MSO}(\Sigma, -)$.

A rational expression α of $\mathbb{TR}$ is called *star-connected* if iteration occurs over sets of connected traces only, i.e., for any subexpression β^* of α, $L(\beta)$ is a set of connected traces. By c-$\mathcal{RAT}_{\mathbb{TR}}$, we denote the set of rational languages that arise from star-connected rational expressions of $\mathbb{TR}$. While, in general, $\mathcal{REC}_{\mathbb{TR}}$ is strictly contained in $\mathcal{RAT}_{\mathbb{TR}}$ [Och95], we obtain equivalence if we restrict to star-connected rational expressions.

Theorem 2.4.2 ([Och95])

$$\mathcal{REC}_{\mathbb{TR}} \;=\; \text{c-}\mathcal{RAT}_{\mathbb{TR}}$$

Let us now clarify what a *regular* trace language is, whose definition defers to recognizability of the corresponding set of linearizations.

Definition 2.4.3 (Regular Trace Language)
A set $L \subseteq \mathbb{TR}(\tilde{\Sigma})$ is called a *regular trace language* (over $\tilde{\Sigma}$) if $Lin(L)$ is a regular word language over Σ, i.e., $Lin(L) \in \mathcal{REC}_{\mathbb{W}(\Sigma)}$.

The class of regular trace languages over $\tilde{\Sigma}$ is denoted by $\mathcal{R}_{\mathbb{TR}(\tilde{\Sigma})}$ or, in accordance with our convention, simply by $\mathcal{R}_{\mathbb{TR}}$.

Another characterization of trace languages is based on the concept of inference, which takes into consideration the distributed nature of a system. Formally, we require a language to be closed under some inference operator $\vdash_{\tilde{\Sigma}}$. Namely, given a set $L \subseteq \mathbb{TR}(\tilde{\Sigma})$ and a trace $T \in \mathbb{TR}(\tilde{\Sigma})$, we write $L \vdash_{\tilde{\Sigma}} T$ if the following holds:

$$\forall i \in [K] : \ \exists T' \in L : \ T' \upharpoonright i = T \upharpoonright i$$

We are now prepared to define what we mean by a *product language*.

Definition 2.4.4 ((Weak) Product Trace Language, cf. [Thi95])
A set $L \subseteq \mathbb{TR}(\tilde{\Sigma})$ is called a *weak product (trace) language* (over $\tilde{\Sigma}$) if, for any $T \in \mathbb{TR}(\tilde{\Sigma})$, $L \vdash_{\tilde{\Sigma}} T$ implies $T \in L$. A trace language $L \subseteq \mathbb{TR}(\tilde{\Sigma})$ is called a *product (trace) language* (over $\tilde{\Sigma}$) if it is the finite union of weak product trace languages.

We denote by $\mathcal{P}^0_{\mathbb{TR}(\tilde{\Sigma})}$ and $\mathcal{P}_{\mathbb{TR}(\tilde{\Sigma})}$ the classes of weak product trace languages and product trace languages, respectively, and just write $\mathcal{P}^0_{\mathbb{TR}}$ and $\mathcal{P}_{\mathbb{TR}}$ if it seems more convenient.

Example 2.4.5 Suppose $\tilde{\Sigma} = (\{a, b\}, \{b, c\})$ and let L consist of all those traces $(E, \lhd, \lambda) \in \mathbb{TR}(\tilde{\Sigma})$ such that there is $e, e' \in E$ with $\lambda(e) = a$, $\lambda(e') = c$, $e \not\leq e'$, and $e' \not\leq e$. Then, though it is a regular trace language, L is not contained in $\mathcal{P}^0_{\mathbb{TR}(\tilde{\Sigma})}$ (we will see below that it is not even a product language). Because if we suppose the trace $T_1 \in L$ to be given by its projections $T_1 \upharpoonright 1 = ab$ and $T_1 \upharpoonright 2 = cb$, while $T_2 \in L$ shall be given by $T_2 \upharpoonright 1 = ba$ and $T_2 \upharpoonright 2 = bc$, then we have both $\{T_1, T_2\} \vdash_{\tilde{\Sigma}} abc \notin L$ (witnessed by $T_1 \upharpoonright 1$ and $T_2 \upharpoonright 2$) and $\{T_1, T_2\} \vdash_{\tilde{\Sigma}} cba \notin L$ (witnessed by $T_1 \upharpoonright 2$ and $T_2 \upharpoonright 1$), which contradicts the definition of a weak product language.

Let us bring together the concepts of regularity and product behavior.

Definition 2.4.6 (Regular Product Trace Language)
A trace language $L \subseteq \mathbb{TR}(\tilde{\Sigma})$ is called a *weak regular product (trace) language* (over $\tilde{\Sigma}$) if both $L \in \mathcal{R}_{\mathbb{TR}(\tilde{\Sigma})}$ and $L \in \mathcal{P}^0_{\mathbb{TR}(\tilde{\Sigma})}$. Moreover, it is said to be a *regular product (trace) language* (over $\tilde{\Sigma}$) if it is the finite union of weak regular product languages.

The corresponding language classes are denoted by $\mathcal{RP}^0_{\mathbb{TR}(\tilde{\Sigma})}$ and, respectively, $\mathcal{RP}_{\mathbb{TR}(\tilde{\Sigma})}$.

2.4.2 Automata for Traces

The distributed nature of traces asks for likewise distributed automata models, which preferably cover some of the classes proposed in the previous subsection. Let us start with asynchronous automata, the most general version of automata for traces that we consider.

Definition 2.4.7 (Asynchronous Automaton [Zie87])
An *asynchronous automaton* over $\tilde{\Sigma}$ is a structure $\mathcal{A} = ((S_i)_{i\in[K]}, (\Delta_a)_{a\in\Sigma}, \overline{s}^{in}, F)$ where

- for each $i \in [K]$, S_i is a nonempty finite set of (i-)*local states*,
- for each $a \in \Sigma$, $\Delta_a \subseteq S_a \times S_a$ is the set of (a-)*synchronizing transitions* where $S_a := \{\overline{s} \in \prod_{i\in[K]}(S_i \uplus \{*\}) \mid \text{for any } i \in [K],\ \overline{s}[i] = * \text{ iff } i \notin loc(a)\}$,
- $\overline{s}^{in} \in \prod_{i\in[K]} S_i$ is the *global initial state*, and
- $F \subseteq \prod_{i\in[K]} S_i$ is the set of *global final states.*

Example 2.4.8 An asynchronous automaton over $(\{a,b\},\{b,c\})$ is illustrated in Figure 2.5 where b-synchronizing transitions are joint by dashed lines. Thus, Δ_b contains both $((s_0,t_0),(s_0,t_0))$ and $((s_1,t_1),(s_1,t_1))$, while Δ_a is given by $\{((s_0,*),(s_0,*)),((s_0,*),(s_1,*)),((s_1,*),(s_1,*))\}$ and, accordingly, Δ_c is given by $\{((*,t_0),(*,t_0)),((*,t_0),(*,t_1)),((*,t_1),(*,t_1))\}$.

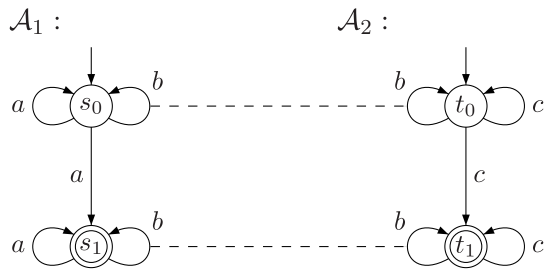
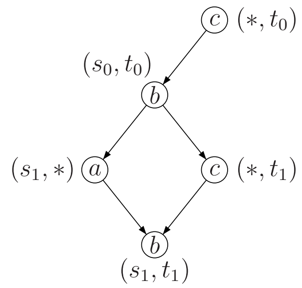


Figure 2.5: An asynchronous automaton over $(\{a,b\},\{b,c\})$

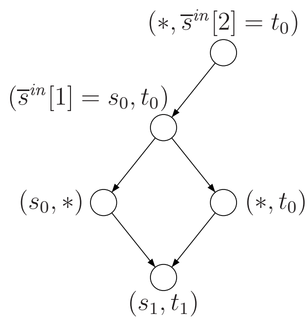
Let $\mathcal{A} = ((S_i)_{i\in[K]}, (\Delta_a)_{a\in\Sigma}, \overline{s}^{in}, F)$ be an asynchronous automaton and let $T = (E, \lhd, \lambda) \in \mathbb{TR}$ be a trace. For a mapping $r : E \to \bigcup_{a\in\Sigma} S_a$ such that, for each $e \in E$, $r(e) \in S_{\lambda(e)}$, we define another mapping $r^- : E \to \bigcup_{a\in\Sigma} S_a$ by

$$r^-(e)[i] = \begin{cases} \overline{s}^{in}[i] & \text{if } e \in E_i \text{ and } e = \mathit{first}(T\restriction i) \\ r(\mathit{last}((T\downarrow e)\restriction i))[i] & \text{if } e \in E_i \text{ and } e \neq \mathit{first}(T\restriction i) \\ * & \text{if } e \notin E_i \end{cases}$$

A *run* of $\mathcal{A}$ on T is a mapping $r : E \to \bigcup_{a\in\Sigma} S_a$ with $r(e) \in S_{\lambda(e)}$ (for each $e \in E$) such that, for any $e \in E$, $(r^-(e), r(e)) \in \Delta_{\lambda(e)}$. For $i \in [K]$, let f_i denote $\overline{s}^{in}[i]$ if $E_i = \emptyset$ and, otherwise, let f_i denote $r(last(T \upharpoonright i))[i]$. We call r *accepting* if $(f_i)_{i\in[K]} \in F$. The *language* of $\mathcal{A}$, $\{T \in \mathbb{TR} \mid$ there is an accepting run of $\mathcal{A}$ on $T\}$, is denoted by $L(\mathcal{A})$.



(a) a run r



(b) the mapping r^-

Figure 2.6: An accepting run of an asynchronous automaton

Example 2.4.9 An accepting run r of the asynchronous automaton $\mathcal{A}$ from Figure 2.5 on the trace of Figure 2.4 on page 28 is illustrated in Figure 2.6 (a) where the structure aside reflects the corresponding mapping r^-. One verifies that the language of $\mathcal{A}$ is the set of traces $(E, \lhd, \lambda) \in \mathbb{TR}(\{a, b\}, \{b, c\})$ such that there is $e, e' \in E$ with $\lambda(e) = a$, $\lambda(e') = c$, $e \not\leq e'$, and $e' \not\leq e$.

By $\mathcal{AA}(\tilde{\Sigma})$ (or simply $\mathcal{AA}$), we denote the class of trace languages that are the language of some asynchronous automaton over $\tilde{\Sigma}$.

Theorem 2.4.10 ([Zie87])

$$\mathcal{REC}_{\mathbb{TR}} \;=\; \mathcal{R}_{\mathbb{TR}} \;=\; \mathcal{AA}$$

Zielonka even shows that, for any asynchronous automaton, there is an equivalent deterministic one, whose definition is omitted here. Basically, for any trace T, a deterministic asynchronous automaton allows at most one run on T.

Theorem 2.3.3, which states that any regular word language can be defined in MSO logic and, vice versa, any MSO sentence over words constitutes a regular word language, carries over to the setting of traces, no matter whether we deal with the underlying partial order or their Hasse diagrams.

Theorem 2.4.11 (cf. [Tho90, Ebi95])

$$\mathcal{EMSO}_{\mathbb{TR}} = \mathcal{MSO}_{\mathbb{TR}} = \mathcal{AA} = \mathcal{MSO}[\leq]_{\mathbb{TR}} = \mathcal{EMSO}[\leq]_{\mathbb{TR}}$$

Let us compare the expressiveness of some automata models for traces.

Corollary 2.4.12

$$1\text{-}\mathcal{GA}_{\mathbb{TR}} = \mathcal{GA}_{\mathbb{TR}} = \mathcal{AA}$$

Proof Analogously to the word case, the second equality follows from Theorem 2.2.13 and Theorem 2.4.11. Moreover, one can easily reduce a graph acceptor of radius R to a graph acceptor of radius 1 involving a blow-up in the number of states [Tho96]. □

Applying Lemma 2.2.12 to traces, we cannot simply remove the concept of occurrence constraints from graph acceptors without losing expressiveness. However, it turns out that the restriction to weak product languages allows us to abstract away from occurrence constraints.

Lemma 2.4.13 For any $L \in \mathcal{P}^0_{\mathbb{TR}}$, we have

$$L \in \mathcal{GA}^-_{\mathbb{TR}} \text{ iff } L \in \mathcal{GA}_{\mathbb{TR}}.$$

The proof is a simple variant of the more general one for Lemma 4.4.13, which, in turn, relies on Corollary 4.4.11. Basically, a graph acceptor, according to Corollary 2.4.12, is reduced towards spheres with radius 1. Afterwards, we construct a special kind of *product automaton* (as it is defined below), which implements a threshold counting procedure to simulate the occurrence constraints and subsequently allows an easy transformation into a graph acceptor without occurrence constraints.

So let us turn to (weak) regular product languages and try to find a suitable automata model that is weaker than the asynchronous one and generates a regular product language in a natural manner. Synchronizing transitions turn out to be essential to recognize the language of the automaton from Figure 2.5 on page 30. Certain product automata lack such a possibility and operate more autonomously than asynchronous automata do. Though product automata are similar to asynchronous automata, they are strictly less expressive.

Definition 2.4.14 (Product Automaton)
A *product automaton* over $\tilde{\Sigma}$ is a structure $\mathcal{A} = ((\mathcal{A}_i)_{i\in[K]}, \overline{s}^{in}, F)$ such that

- for each $i \in [K]$, $\mathcal{A}_i$ is a pair (S_i, Δ_i) where

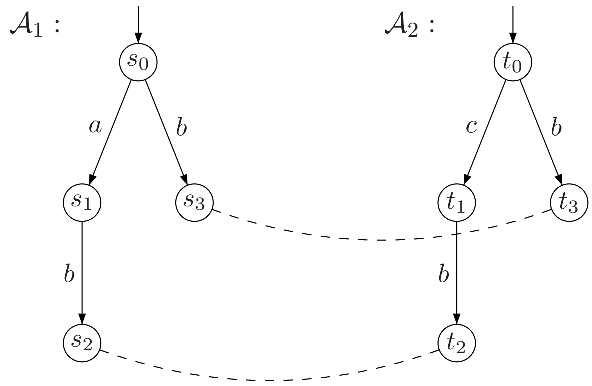


Figure 2.7: A product automaton over $(\{a, b\}, \{b, c\})$

- S_i is a nonempty finite set of (i-)*local states* and
- $\Delta_i \subseteq S_i \times \Sigma_i \times S_i$ is the set of (i-)*local transitions*,

– $\overline{s}^{in} \in \prod_{i \in [K]} S_i$ is the *global initial state*, and

– $F \subseteq \prod_{i \in [K]} S_i$ is the set of *global final states*.

A simple finite product automaton over $(\{a, b\}, \{b, c\})$ is illustrated in Figure 2.7 where global final states are depicted by dashed lines.

The behavior of a product automaton is quite similar to the one of an asynchronous automaton but more autonomous. So let $\mathcal{A} = ((\mathcal{A}_i)_{i \in [K]}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be a product automaton over $\tilde{\Sigma}$ (again, set S_a to be $\{\overline{s} \in \prod_{i \in [K]}(S_i \uplus \{*\}) \mid$ for any $i \in [K]$, $\overline{s}[i] = *$ iff $i \notin loc(a)\}$) and suppose $T = (E, \lhd, \lambda)$ to be a trace over $\tilde{\Sigma}$. A *run* of $\mathcal{A}$ on T is a mapping $r : E \to \bigcup_{a \in \Sigma} S_a$ with $r(e) \in S_{\lambda(e)}$ (for each $e \in E$) such that, for any $e \in E$ and any $i \in loc(\lambda(e))$, $(r^{-}(e)[i], r(e)[i]) \in \Delta_i$ (where r^{-} is defined as above). It remains to clarify when r is accepting. For $i \in [K]$, let f_i denote $\overline{s}^{in}[i]$ if $E_i = \emptyset$. Otherwise, let f_i denote $r(last(T \upharpoonright i))[i]$. We call r *accepting* if $(f_i)_{i \in [K]} \in F$. The *language* of $\mathcal{A}$, $\{T \in \mathbb{TR} \mid$ there is an accepting run of $\mathcal{A}$ on $T\}$, is denoted by $L(\mathcal{A})$.

By $\mathcal{PA}(\tilde{\Sigma})$ ($\mathcal{PA}$ if $\tilde{\Sigma}$ is clear from the context), we denote the set of trace languages that is determined by the class of product automata over $\tilde{\Sigma}$.

Lemma 2.4.15 If there is $i, j \in [K]$ such that $\Sigma_i \cap \Sigma_j \neq \emptyset$ and neither $\Sigma_i \subseteq \Sigma_j$ nor $\Sigma_j \subseteq \Sigma_i$, then

$$\mathcal{PA} \subsetneqq \mathcal{AA}.$$

Proof Suppose $\tilde{\Sigma} = (\{a,b\},\{b,c\})$ (i.e., $K = 2$) and L consists of those traces $(E, \lhd, \lambda) \in \mathbb{TR}(\tilde{\Sigma})$ such that there is $e, e' \in E$ with $\lambda(e) = a$, $\lambda(e') = c$, $e \not\leq e'$, and $e' \not\leq e$. As mentioned above, L is recognized by the asynchronous automaton illustrated in Figure 2.5 on page 30. Now suppose there is a product automaton $\mathcal{A} = ((\mathcal{A}_i)_{i\in[2]}, \overline{s}^{in}, F)$ over $\tilde{\Sigma}$ recognizing L. For $m, n \in \mathbb{N}$, consider the trace $T(m,n) \in \mathbb{TR}(\tilde{\Sigma})$, which is given by its linearization $b^m acb^n$. If m and n are sufficiently large, $\mathcal{A}_1$, in a successful run of $\mathcal{A}$ on $T(m,n)$, goes through a cycle, say, of length i (≥ 1), to read the first m b's of the word $b^m ab^n$. Similarly, $\mathcal{A}_2$, in the same run, goes through a cycle, say of length j (≥ 1), to read the second n b's of $b^m cb^n$. But then, we can easily construct an accepting run of $\mathcal{A}$ on $b^m cb^{i\cdot j} ab^n \in \mathbb{TR}(\tilde{\Sigma}) \setminus L$, which contradicts the premise. $\square$

Theorem 2.4.16 ([Thi95])

$$\mathcal{RP}_{\mathbb{TR}} \;=\; \mathcal{PA}$$

Note that $\mathcal{RP}^0_{\mathbb{TR}}$ corresponds to the special form of product automata where the set of global final states is the cartesian product of local state spaces rather than a subset of the set of global states so that, actually, we deal with a local acceptance condition. Such correspondence will be studied in more detail in the framework of message sequence charts.

2.5 Pictures and Grids

An important class of graphs is provided by *pictures*. Many results on pictures will be used to achieve the corresponding results in the framework of message sequence charts. Once more, pictures are a special case of graphs. However, while the node labeling is arbitrary, an edge of a picture is labeled with either 1 or 2. So let Σ be an alphabet in the following and, given $n \in \mathbb{N}_{\geq 1}$, $[n]$ denote the set $\{1, \ldots, n\}$.

Definition 2.5.1 (Picture)
A *picture* over Σ is a structure $([n]\times[m], S_1, S_2, \lambda) \in \mathbb{DG}_H(\Sigma, \{1,2\})$ with $n, m \in \mathbb{N}_{\geq 1}$ where $S_1, S_2 \subseteq ([n]\times[m])^2$ contain the pairs $((i,j),(i+1,j)) \in ([n]\times[m])^2$ and $((i,j),(i,j+1)) \in ([n]\times[m])^2$, respectively, and λ is a mapping $[n]\times[m] \to \Sigma$.

The set of pictures over Σ is denoted by $\mathbb{P}(\Sigma)$ ($\mathbb{P}$ if Σ is clear from the context). Note that, in the context of pictures, we use S_1 and S_2 rather than $\lhd_1$ and $\lhd_2$ to denote the edge relations, respectively, because this is more common. For example, Figure 2.8 on the facing page shows a picture over $\{a,b,c\}$ with $n = 3$

rows and $m = 8$ columns. In addition, the vertical arrows are labeled with 1 while the horizontal ones are labeled with 2, which is omitted here for the sake of clarity.

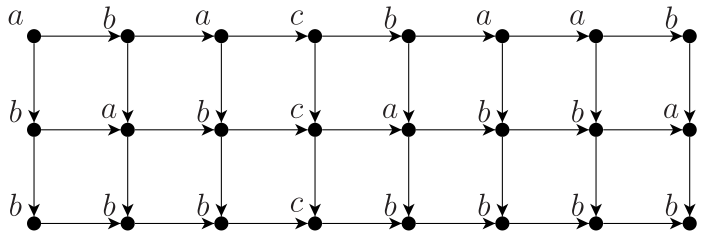


Figure 2.8: A picture over $\{a, b, c\}$

Though *tiling systems*, where a tiling is a square of length two rather than a graph around a center (see [GRST96] for an overview), is arguably the more natural automata model for pictures, we stick to the familiar terrain of graph acceptors. As with traces, graph acceptors yield different results than the general case, when they are applied to pictures rather than arbitrary graphs.

Lemma 2.5.2 ([Tho96])

$$1\text{-}\mathcal{GA}^{-}_{\mathbb{P}(\Sigma)} \;=\; \mathcal{GA}_{\mathbb{P}(\Sigma)}$$

Again, the proof is a simple reduction involving a blow-up in the number of states of the graph acceptor.

Theorem 2.5.3 ([Tho96]) In general, $\mathcal{FO}[\leq]_{\mathbb{P}(\Sigma)}$ and $\mathcal{EMSO}_{\mathbb{P}(\Sigma)}$ are incomparable wrt. inclusion.

Proof Let $\Sigma = \{a, b, c\}$. The set L of pictures over Σ that consist of one single column of even length is $\mathrm{EMSO}_{\mathbb{P}(\Sigma)}$-definable. But as the set of words over Σ of even length is not $\mathrm{FO}[\leq]_{\mathbb{W}(\Sigma)}$-definable, L cannot be $\mathrm{FO}[\leq]_{\mathbb{P}(\Sigma)}$-definable.

Conversely, assume $L \subseteq \mathbb{P}(\Sigma)$ to be the set of pictures over Σ that can be considered as the *concatenation* GCH of some picture $C \in \mathbb{P}(\{c\})$ consisting of one single c-labeled column and pictures $G, H \in \mathbb{P}(\{a, b\})$ such that the sets of different column labelings of G and H coincide. A picture that belongs to L is depicted in Figure 2.8. Its unique division into G, C, and H as postulated above and illustrated in Figure 2.9 on the following page gives rise to the set of column words $\{abb, bab\}$, which represents both G and H. In fact, L is

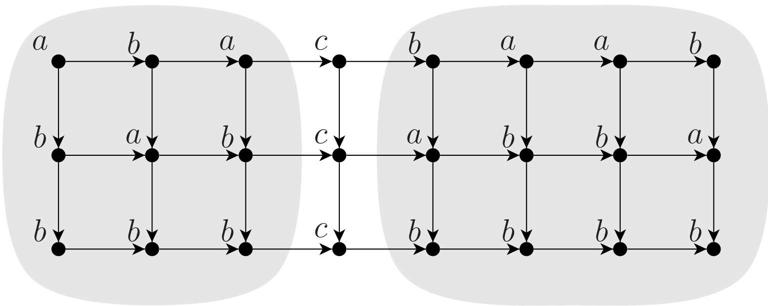


Figure 2.9: Dividing a picture over $\{a, b, c\}$

$\mathrm{FO}[\leq]_{\mathbb{P}(\Sigma)}$-definable. A corresponding sentence just has to require that, for any event x on the first row, there has to be a suitable counterpart y that is also located on the first row, but on the opposite side of the c-labeled column. Moreover, the columns below x and y have to coincide. The latter can be easily formalized using the predicates $\leq_1$ and $\leq_2$ (with the obvious meaning, i.e., $\leq_2$ stands for proceeding from left to right), which, in turn, are definable only in terms of $\leq$. However, L cannot be $\mathrm{EMSO}_{\mathbb{P}(\Sigma)}$-definable. Because, if we suppose $L \in \mathcal{EMSO}_{\mathbb{P}(\Sigma)}$, then, according to Theorem 2.2.13 and Lemma 2.5.2, there is a graph acceptor $\mathcal{B} = (Q, R, \mathfrak{S}, Occ)$ over $(\Sigma, \{1, 2\})$ such that $R = 1$, Occ is logically equivalent to true, and $L_{\mathbb{P}(\Sigma)}(\mathcal{B}) = L$. In any accepting run of $\mathcal{B}$ on a picture GCH, all the information carried from G to H is already present in the sequence of states associated to the single column C. For a given column length n, the number of those state sequences is $|Q|^n$. In contrast, the number of distinct nonempty sets of words over $\{a, b\}$ of length n is $2^{2^n} - 1$ and, therefore, exceeds $|Q|^n$ for sufficiently large n. So we can find an accepting run on $G'CH'$ for some $G', H' \in \mathbb{P}(\{a, b\})$ that induce different sets of column words. $\square$

A special case of a picture is given if Σ is a singleton, which allows us to omit the labeling function. We then rather speak of a *grid*. Given $n, m \in \mathbb{N}_{\geq 1}$, the (n, m)-*grid* (with n rows and m columns) is the structure $G(n, m) := ([n] \times [m], S_1, S_2) \in \mathbb{DG}_H(-, \{1, 2\})$ where, as in the picture case, $S_1, S_2 \subseteq ([n] \times [m])^2$ contain the pairs $((i, j), (i + 1, j)) \in ([n] \times [m])^2$ and $((i, j), (i, j + 1)) \in ([n] \times [m])^2$, respectively. By $\mathbb{GR}$, we denote the set of all grids. The (3,5)-grid is depicted in Figure 2.10 on the next page. A relation $R \subseteq \mathbb{N}_{\geq 1} \times \mathbb{N}_{\geq 1}$ may be represented by the grid language $\{G(n, m) \mid (n, m) \in R\}$. As a unary function $f : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ can be considered as a binary relation, we define the *grid language* $\mathcal{G}(f)$ of f to be the set $\{G(n, f(n)) \mid n \in \mathbb{N}_{\geq 1}\}$.

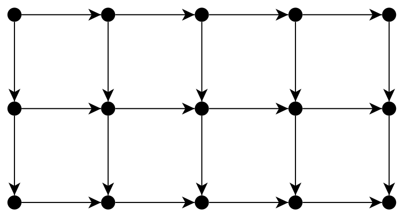


Figure 2.10: The (3,5)-grid

By means of grids, Matz and Thomas showed that quantifier alternation of second-order variables in MSO logic over graphs forms an infinite hierarchy.

Theorem 2.5.4 ([MT97]) The monadic quantifier-alternation hierarchy over $\mathbb{GR}$ is infinite.

The next result directly follows from Lemma 2.5.2. Again, the proof is a simple reduction involving a blow-up in the number of states of the graph acceptor.

Corollary 2.5.5

$$1\text{-}\mathcal{GA}^{-}_{\mathbb{GR}} = \mathcal{GA}_{\mathbb{GR}}$$

Though, wrt. grids, already 1-spheres suffice to give graph acceptors the full expressive power, grids are the starting point to prove that, in general, one cannot restrict to 1-spheres (recall Lemma 2.2.11). This fact is witnessed by the set L_n of n-*supergrids* for some $n \geq 4$ (cf. [Tho96]). From a grid, we obtain the corresponding n-supergrid if any edge is replaced by a sequence of n new edges. Such a sequence is called a *superedge*. Relative to $\mathbb{DG}(-, \{1, 2\})$, L_n can be recognized by some graph acceptor equipped with $2n$-spheres. But now suppose there is a graph acceptor $\mathcal{B}$ with radius 1 such that $L_{\mathbb{DG}}(\mathcal{B}) = L_n$ and consider ρ to be an accepting run of $\mathcal{B}$ on some $G \in L_n$. If G is chosen to be large enough, then ρ might exhibit two occurrences of the same 1-sphere whose nodes do not touch the end of a superedge and are not related in some way wrt. the partial order induced by G. Moreover, suppose the nodes of G that belong to the corresponding two sphere centers to be e_1 and e_2 and assume that their (only) outgoing edges lead to e'_1 and e'_2, respectively. From G, we obtain another grid if we remove the edges (e_1, e'_1) and (e_2, e'_2) and, instead, add (e_1, e'_2) and (e_2, e'_1). The resulting grid, though it is contained in $\mathbb{DG}$ and in $L_{\mathbb{DG}}(\mathcal{B})$, is no longer a supergrid.

Chapter 3

Message Sequence Charts

One of the first papers aiming at giving message sequence charts (MSCs) a formal semantics was [Mau96], which described the behavior of an MSC in a process algebra. To make MSCs accessible to automata theory and logics, however, it pays off to consider an MSC to be a partial order [KL98] or a graph [BAL97], whose entities are handled by automata more naturally. Let us now formally introduce MSCs, which can be seamlessly embedded into the framework provided in Chapter 2.

3.1 Message Sequence Charts

Forthcoming definitions are all made wrt. a fixed finite set P of at least two *processes*. (Note that, however, in proofs, we sometimes silently assume the existence of more than two processes.) We denote by $Ch(P)$ the set $\{(p,q) \mid p,q \in P,\ p \neq q\}$ of reliable FIFO *channels*. Thus, a message exchange is allowed between distinct processes only. Let $Act^!(P)$ denote the set $\{p!q \mid (p,q) \in Ch(P)\}$ of *send actions* while $Act^?(P)$ denotes the set $\{q?p \mid (p,q) \in Ch(P)\}$ of *receive actions*. Hereby, $p!q$ and $q?p$ are to be read as p *sends a message to* q and q *receives a message from* p, respectively. They are related in the sense that they will label communicating events of an MSC, which are joint by a message arrow in its graphical representation. Accordingly, we set $Com(P) := \{(p!q, q?p) \mid (p,q) \in Ch(P)\}$ and, for some distinct p,q, $Com(P)_{(p,q)} := \{(p!q, q?p)\}$. Observe that an action $p\theta q$ ($\theta \in \{!, ?\}$) is performed by process p, which is indicated by $P(p\theta q) = p$. We let $Act(P)$ stand for the union of $Act^!(P)$ and $Act^?(P)$ and, for $p \in P$, set $Act(P)_p$ to be the set $\{\sigma \in Act(P) \mid P(\sigma) = p\}$. Moreover, we use P_{c} as a shorthand for the set $P \uplus \{\mathrm{c}\}$ (the symbol c will be subsequently used to label message arrows in an MSC, while a process will label the successor relation of the corresponding process line). If P is clear from the context (which is often the case), we take the liberty of omitting the reference to P and just write Ch, $Act^!$, $Act^?$, Act, and Com, for example.

An MSC will be defined stepwise, starting with *partial MSCs*, where some events may lack a suitable communication partner. This view is concretized towards *left-closed MSCs*, which allow at most unmatched send events, while, finally, a *basic MSC* describes a complete behavior without any open events.

Definition 3.1.1 (Partial Message Sequence Chart)
A *partial message sequence chart* (over P) is a graph $(E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda) \in \mathbb{DG}(Act, P_c)$ such that

- $\lhd_p$ is the covering relation of some total order on $E_p := \lambda^{-1}(Act_p)$ (recall that this total order is then denoted by $\leq_p$),
- $\lhd_c \subseteq E \times E$ such that, for any $(e_1, e_1'), (e_2, e_2') \in \lhd_c$,
  - $(\lambda(e_1), \lambda(e_1')) \in Com$ and
  - if $(\lambda(e_1), \lambda(e_1')) = (\lambda(e_2), \lambda(e_2')) \in Com_{(p,q)}$ for some $(p, q) \in Ch$, then $e_1 <_p e_2$ iff $e_1' <_q e_2'$.

Recall that λ is a labeling function of type $E \to Act$ and $\lhd^* = (\lhd_c \cup \bigcup_{p \in P} \lhd_p)^*$ is required to be a partial order. Moreover, E is a nonempty finite set, whose elements are called *events*. Events on one and the same process line are totally ordered and events on distinct process lines that are immediately concerned with each other (wrt. $\lhd_c$) are labeled with actions related by Com. Given $e \in E$, $P(e)$ will subsequently serve as a shorthand for $P(\lambda(e))$. The definition of a partial MSC guarantees that completed message transfers, which correspond to elements from $\lhd_c$, are processed along a FIFO architecture. However, there might still be unmatched events. So let us identify events of a partial MSC $M = (E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda)$ that either have no communication partner yet or just lack a link to an existing partner and let accordingly U_M denote the set $\{e \in E \mid$ there is no $e' \in E$ such that $e \lhd_c e'$ or $e' \lhd_c e\}$ of *unmatched events* of M.

Definition 3.1.2 (Left-Closed Partial Message Sequence Chart)
A partial MSC M is called *left-closed* if $\lambda(U_M) \subseteq Act^!$.

In other words, a left-closed partial MSC has no unmatched receive events. In an MSC, finally, any event is part of a full message exchange:

Definition 3.1.3 (Message Sequence Chart)
A *message sequence chart* is a partial MSC M such that $U_M = \emptyset$.

The set of partial MSCs over P is denoted by $\mathrm{p}\mathbb{MSC}(P)$, the set of MSCs by $\mathbb{MSC}(P)$. The members of $\mathbb{MSC}(P)$ are often called *basic* MSCs. When P will be clear from the context, a corresponding reference is often omitted. Recall that $\mathrm{p}\mathbb{MSC}(P) \subseteq \mathbb{DG}(Act(P), P_{\mathrm{c}})$. An *MSC language* over P is a subset of $\mathbb{MSC}(P)$. The class $2^{\mathbb{MSC}(P)}$ of all those MSC languages over P is denoted by $\mathcal{MSC}(P)$ or just $\mathcal{MSC}$.

To be able to apply the theory of graph acceptors and, in particular, Theorem 2.2.13, the following remark will prove important.

Remark 3.1.4 $\mathbb{MSC}$ has bounded degree.

In fact, the degree of $\mathbb{MSC}$ is bounded by 3.

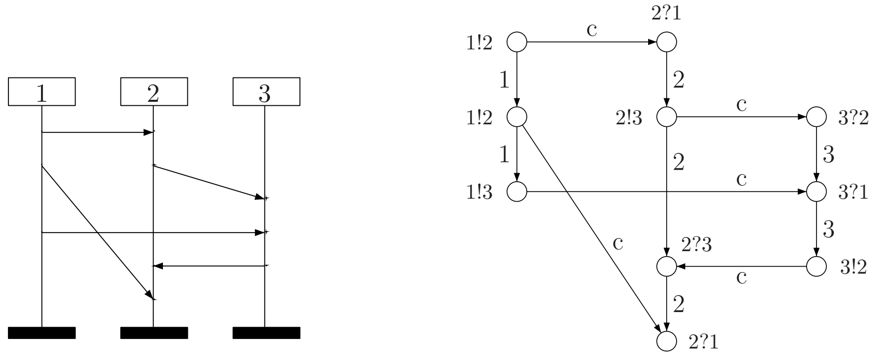
Given a partial MSC $M = (E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda)$, we might argue that some pair of unmatched events can be combined towards a complete message. Following this idea, we define a relation $\widehat{\lhd_{\mathrm{c}}} \subseteq E \times E$, which accordingly relates events from U_M. Intuitively, they are queued into a FIFO channel. For $e, e' \in E$, we formally write $e \mathrel{\widehat{\lhd_{\mathrm{c}}}} e'$ if, for some channel $(p, q) \in Ch$,

- $\{e, e'\} \subseteq U_M$,
- $(\lambda(e), \lambda(e')) \in Com_{(p,q)}$,
- $|\{e'' \in E_p \cap U_M \mid e'' \leq_p e \text{ and } \lambda(e'') = p!q\}| =$
  $|\{e'' \in E_q \cap U_M \mid e'' \leq_q e' \text{ and } \lambda(e'') = q?p\}|$, and
- for any $(e_1, e_1') \in \lhd_{\mathrm{c}}$ with $(\lambda(e_1), \lambda(e_1')) \in Com_{(p,q)}$, $e <_p e_1$ iff $e' <_q e_1'$.

Observe that, if $(\lhd \cup \widehat{\lhd_{\mathrm{c}}})^*$ is antisymmetric, then $(E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}} \cup \widehat{\lhd_{\mathrm{c}}}, \lambda)$ is a partial MSC, which can be understood as the *adjustment* of M along the FIFO architecture. In that case, we say that M *represents* the partial MSC $(E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}} \cup \widehat{\lhd_{\mathrm{c}}}, \lambda)$.

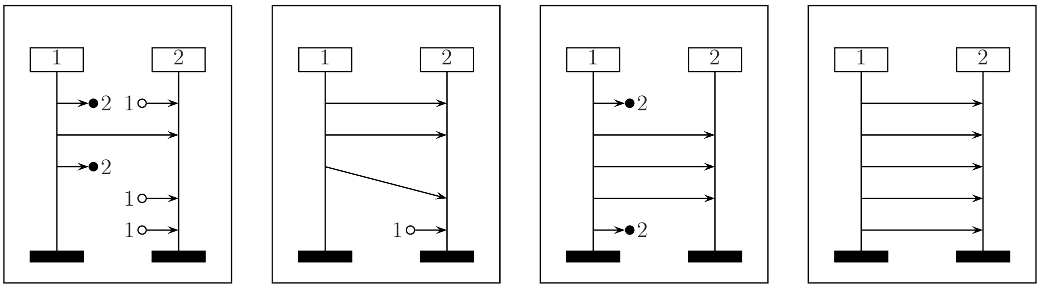
Summarizing, we model an MSC as a graph, adopting the view taken in [Mad01, LL95, BAL97] rather than considering partial orders [HMKT00a, Mor02, Kus03]. As we will discuss in more detail, this does not affect our main results. Note that, though we consider partial MSCs with possibly unmatched events, a system description or an implementation of a system is designed for complete message transfer in this thesis, i.e., partial MSCs will be combined here towards basic MSCs. A method to cope with messages that are *found* or definitely *lost*, which are supplied by the MSC standard, is provided in [BLL02].

An MSC is depicted in part (b) of Figure 3.1 on the following page. However, to illustrate an MSC, we mostly represent it by a diagram such as shown in



(a) (b)

Figure 3.1: An MSC



(a) (b) (c) (d)

Figure 3.2: Three sample partial MSCs

Figure 3.1 (a), which is more intuitive and provides enough information to infer the corresponding graph. This example shows that it would be too restrictive if we confined ourselves to graphs from $\mathbb{DG}_H(Act, P_c)$, as the edge representing the second message from process 1 to process 2 is already implicitly present. If, in general, we deal with partial MSCs, we indicate an unmatched event as illustrated by Figure 3.2 showing two partial MSCs that are not left-closed (a,b), a left-closed partial MSC that is not an MSC (c), and an MSC (d). Hereby, the labeling of an unmatched event indicates the desired communication partner. Recall that the partial MSC from Figure 3.2 (a) represents the one from Figure 3.2 (b).

For a partial MSC $M = (E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda)$ and a process $p \in P$, the *projection* of M onto p, denoted by $M \upharpoonright p$, is defined to be the empty word if $E_p = \emptyset$ and, otherwise, to be $M \upharpoonright (Act_p, \{p\})$, which is identical to $(E_p, \lhd_p, \lambda_{|E_p}) \in \mathbb{W}(Act_p)$. Note that an MSC M is uniquely determined by the collection $(M \upharpoonright p)_{p \in P}$ of its projections, which is obviously not the case if we consider partial MSCs.

Given two partial MSCs $M = (E, \{\lhd_p\}_{p\in P}, \lhd_c, \lambda)$ and $M' = (E', \{\lhd'_p\}_{p\in P}, \lhd'_c, \lambda')$, let $M \cdot M' := (E'', \{\lhd''_p\}_{p\in P}, \lhd''_c, \lambda'')$ be the *(asynchronous) concatenation* of M and M' where $E'' = E \uplus E'$, $\lambda'' = \lambda \cup \lambda'$, $\lhd''_c = \lhd_c \cup \lhd'_c$, and, for any $p \in P$, $\lhd''_p = \lhd_p \cup \lhd'_p \cup \{(e,e') \in E_p \times E'_p \mid e = \mathit{last}(M \upharpoonright p) \text{ and } e' = \mathit{first}(M' \upharpoonright p)\}$. Note that asynchronous concatenation is associative. As with traces, we add some unit MSC $\mathbf{1}_{\mathbb{MSC}}$ to $\mathrm{p}\mathbb{MSC}$ and $\mathbb{MSC}$ (which can be assumed to be kind of empty (partial) MSC where any component is the empty set) and obtain the monoids $(\mathrm{p}\mathbb{MSC}, \cdot, \mathbf{1}_{\mathbb{MSC}})$ and $(\mathbb{MSC}, \cdot, \mathbf{1}_{\mathbb{MSC}})$, which we identify with $\mathrm{p}\mathbb{MSC}$ and $\mathbb{MSC}$, respectively. Even if, for simplicity, most definitions (such as message-passing automata) are designed for nonempty MSCs only, we will include them implicitly and it will be clear how to extend definitions to cope with $\mathbf{1}_{\mathbb{MSC}}$ as well.

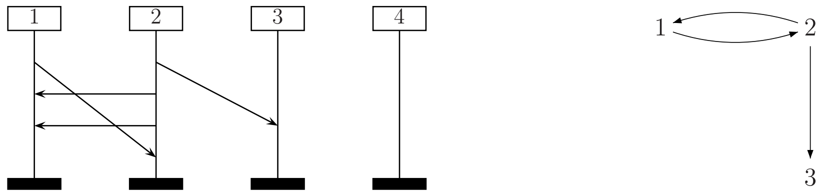
Remark 3.1.5 Both $\mathrm{p}\mathbb{MSC}$ and $\mathbb{MSC}$ are not finitely-generated.

For a partial MSC $M = (E, \{\lhd_p\}_{p\in P}, \lhd_c, \lambda) \in \mathrm{p}\mathbb{MSC}$, we define the *communication graph* of M, denoted by $\mathrm{c}G(M)$, to be the pair $(P(M), \mathit{Arcs})$ where $P(M) := \{P(e) \mid e \in E\}$ and, for any $p, q \in P(M)$, $(p,q) \in \mathit{Arcs}$ if there are $e, e' \in E$ such that $(\lambda(e), \lambda(e')) \in \mathit{Com}_{(p,q)}$ [GKM04]. Remarkably, if M is a basic MSC, then M is connected iff $\mathrm{c}G(M)$ is connected, which does not hold for partial MSCs in general. Figure 3.3 on the following page shows a connected MSC and its communication graph. Note that a basic MSC over at most three processes is always connected, which does not apply to partial MSCs.

3.2 Universal and Existential Bounds

An important subclass of $\mathbb{MSC}$ is identified when we focus on *bounded* MSCs, which cope with systems whose channel capacity is restricted. Those systems turn out to have simpler, more liberal logical characterizations than their unrestricted counterparts and, furthermore, enjoy some nice algorithmic properties (see [Gen04] for an overview). In general, we distinguish two kinds of boundedness. If we require any execution of an MSC (by which we mean a linear extension) to correspond to a fixed channel capacity, we will speak of a *universally*-bounded MSC [HMKT00a]. If, in contrast, we require at least one linearization to fit into the channel restriction, we call an MSC *existentially*-bounded [LM04]. While *regularity* gives rise to universally-bounded MSC languages, an existential bound suffices to ensure decidability of some model-checking problems such as the problem whether an MSO formula satisfies a given *high-level MSC* (see section 3.5) [Mad01, MM01].

Let $B \geq 1$. As we define boundedness in terms of linear extensions of MSCs,



(a) (b)

Figure 3.3: A connected MSC and its communication graph

we first call a word $w \in \mathbb{W}(Act)$ B-bounded if, for any prefix v of w and any $(p,q) \in Ch$, $|v|_{p!q} - |v|_{q?p} \leq B$. An MSC $M \in \mathbb{MSC}$ is called

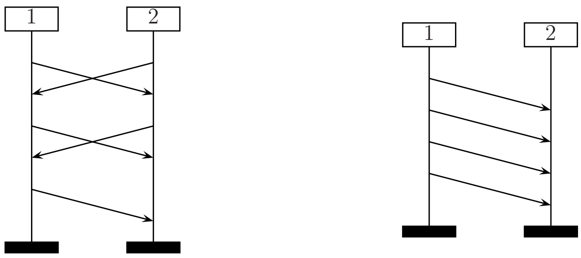
– *universally-B-bounded* ($\forall B$-bounded) if, for any $w \in Lin(M)$, w is B-bounded and

– *existentially-B-bounded* ($\exists B$-bounded) if there is at least one $w \in Lin(M)$ such that w is B-bounded.

In other words, universal boundedness is safe in the sense that any possible execution sequence does not claim more memory than some given upper bound, whereas existential boundedness allows an MSC to be executed even if this does not apply to each of its linear extensions.

Example 3.2.1 The MSC depicted in part (a) of Figure 3.4 on the next page is $\forall 2$-bounded because, at any time of execution, there are at most two messages in each channel: consider process 1, which sends a message to process 2, then receives some message and, again, sends a message to its opponent. At that time, there might be two messages in channel $(1,2)$. However, before process 1 is able to complete its process line by sending a third message, it has to receive a second message from process 2, which, in turn, is required to take at least one message from $(1,2)$. As process 1 can potentially send its second message without awaiting collection of the first one by process 2, the MSC is not $\forall 1$-bounded. The MSC depicted in Figure 3.4 (b) is $\exists 1$-bounded: its linearization $((1!2)(2?1))^4$ makes use of only one location in channel $(1,2)$. Moreover, it is $\forall B$-bounded iff $B \geq 4$.

Given some natural $B \geq 1$, the set of $\forall B$-/$\exists B$-bounded MSCs is denoted by $\mathbb{MSC}_{\forall B}/\mathbb{MSC}_{\exists B}$, respectively. An MSC language $L \subseteq \mathbb{MSC}$ is called $\forall B$-/$\exists B$-bounded if, for any $M \in L$, M is $\forall B$-/$\exists B$-bounded. Moreover, we call L universally-/existentially-bounded ($\forall$-/$\exists$-bounded) if it is $\forall B$-/$\exists B$-bounded for some B.



(a) (b)

Figure 3.4: A $\forall 2$-bounded and an $\exists 1$-bounded MSC

3.3 Relationships to Mazurkiewicz Traces

We recall two approaches to bridge the gap between traces and MSCs. The one is tailored to universally-bounded MSC languages [Kus03] and applies some relabeling to the events of an MSC to obtain a trace, while the other, basically considering several events of an MSC to be an event of a trace, is likewise applicable to unbounded behaviors [Mor02].

3.3.1 The Counting Lemma

In the following, let B be a positive natural. We define $\mathcal{D}_B$ to be the dependence alphabet $(\mathit{Act} \times \{1, \ldots, B\}, D_B)$ where $(\sigma_1, n_1) D_B (\sigma_2, n_2)$ if

- $P(\sigma_1) = P(\sigma_2)$ or
- $(\sigma_1, \sigma_2) \in \mathit{Com} \cup \mathit{Com}^{-1}$ and $n_1 = n_2$.

Note that, though the definitions from this subsection each depend on P, we omit a corresponding reference. Setting Co to be $\{(\sigma, \tau, n) \mid (\sigma, \tau) \in \mathit{Com},\ n \in \{1, \ldots, B\}\}$, let $\tilde{\Sigma}_B$ be the distributed alphabet $(\overline{\mathit{Act}_\gamma})_{\gamma \in P \cup \mathit{Co}}$ where, for $p \in P$, $\overline{\mathit{Act}_p} := \mathit{Act}_p \times \{1, \ldots, B\}$ and, for $(\sigma, \tau, n) \in \mathit{Co}$, $\overline{\mathit{Act}_{(\sigma,\tau,n)}} := \{(\sigma, n), (\tau, n)\}$. For example, if $P = \{1, 2\}$ and $B = 2$,

$$\begin{aligned}\tilde{\Sigma}_B = (\ & \{(1!2,1),(1?2,1),(1!2,2),(1?2,2)\},\\ & \{(2!1,1),(2?1,1),(2!1,2),(2?1,2)\},\\ & \{(1!2,1),(2?1,1)\},\\ & \{(1!2,2),(2?1,2)\}\ ).\end{aligned}$$

Remark 3.3.1 Given $B \geq 1$, $\mathcal{D}_{\tilde{\Sigma}_B} = \mathcal{D}_B$.

To an MSC $M = (E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda) \in \mathbb{MSC}_{\forall B}$, we assign the graph $Tr_B(M) := (E', \lhd', \lambda') \in \mathbb{DG}_H(Act \times \{1, \ldots, B\}, -)$ where $E' = E$, $\lhd'$ is the covering relation of $(\lhd_c \cup \bigcup_{p \in P} \lhd_p)^*$, and, for each $e \in E$, we define $\lambda'(e)$ to be the new labeling $(\lambda(e), |M \Downarrow e|_{\lambda(e)} \mod B)$.

Lemma 3.3.2 ([Kus03]) Let $B \geq 1$. For each MSC $M \in \mathbb{MSC}_{\forall B}$, $Tr_B(M)$ is a trace over $\tilde{\Sigma}_B$.

Note that the mapping $Tr_B : \mathbb{MSC}_{\forall B} \to \mathbb{TR}(\tilde{\Sigma}_B)$ is injective. It is canonically extended towards MSC languages. Thus, involving some relabeling, an MSC language $L \subseteq \mathbb{MSC}_{\forall B}$ can be converted into some trace language $Tr_B(L) \subseteq \mathbb{TR}(\tilde{\Sigma}_B)$.

Example 3.3.3 Taking $M \in \mathbb{MSC}_{\forall 2}$ to be the MSC from Figure 3.4 (a), the trace $Tr_2(M)$ over $\tilde{\Sigma}_2$ is shown in Figure 3.5. Note that, in fact, the labelings of the events e and e', which are incompatible wrt. $\leq$, are independent as they do not occur together in some local alphabet of $\tilde{\Sigma}_2$.

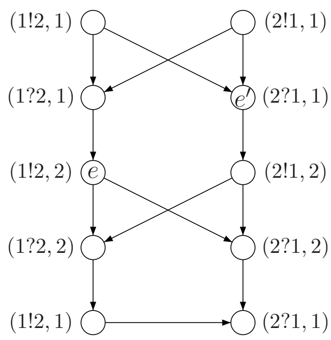
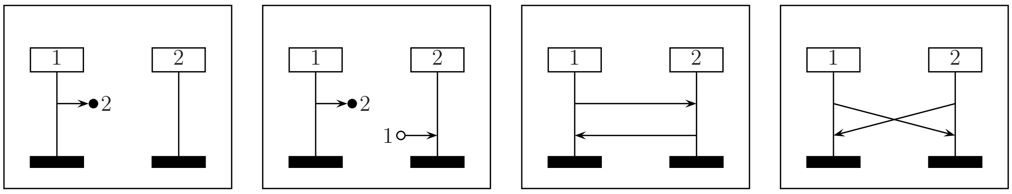


Figure 3.5: The trace of a 2-bounded MSC

Note that, in [Gen04], the above relabeling is applied to existentially-bounded MSCs, too, which gives also rise to Mazurkiewicz traces if we add some edges between events that are actually independent in the MSC.

3.3.2 Prime Partial Message Sequence Charts

For a partial MSC M with labeling function λ, let again $P(M)$ be a shortcut for $\{p \in P \mid \lambda^{-1}(Act_p) \neq \emptyset\}$. A nonempty partial MSC M is called *prime* if $M = M_1 \cdot M_2$ implies $M_1 = \mathbf{1}_{\mathbb{MSC}}$ or $M_2 = \mathbf{1}_{\mathbb{MSC}}$. Consider Figure 3.6 for



(a) (b) (c) (d)

Figure 3.6: Some prime and some non-prime partial MSCs

examples. The partial MSCs from parts (a) and (d) are prime, while the partial MSCs in between are not. For the rest of this subsection, let Γ be a nonempty finite set of prime partial MSCs, which will be the universe of a trace alphabet. The notion of prime partial MSCs, which was first given in [HM00], gives rise to a natural dependence relation based on the distributed alphabet $\tilde{\Sigma}_\Gamma := (\Sigma_p)_{p \in P}$ where, for any $p \in P$, $\Sigma_p = \{M \in \Gamma \mid p \in P(M)\}$. We may accordingly declare prime partial MSCs M and M' independent if $P(M) \cap P(M') = \emptyset$.

Lemma 3.3.4 The morphism $\mathfrak{R}_\Gamma : \mathbb{TR}(\tilde{\Sigma}_\Gamma) \to \langle \Gamma \rangle_{\mathbb{pMSC}}$ that maps a trace over $\tilde{\Sigma}_\Gamma$ with linearization $a_1 \ldots a_n$ onto the partial MSC $a_1 \cdot \ldots \cdot a_n$ is an isomorphism.

The proof is an adaption of a similar one in [Mor02] where the corresponding is shown for basic MSCs rather than partial MSCs.

A basic MSC may correspond to several traces, i.e., there might be distinct traces $T_1, T_2 \in \mathbb{TR}(\tilde{\Sigma}_\Gamma)$ such that $\mathfrak{R}_\Gamma(T_1)$ and $\mathfrak{R}_\Gamma(T_2)$ represent the same MSC. This applies, for example, to the traces from Figure 3.7: though they are different, they represent one and the same basic MSC. However, in our main application of prime partial MSCs in Chapter 5, those traces are not distinguished anyway.

3.4 Message Contents

So far, we did not consider which message is actually sent performing a send event. To enrich our formalism that way, we augment each action with an additional labeling indicating what kind of message is sent or received. Accordingly, an MSC is defined wrt. P and a nonempty finite set of *message contents* Λ. An action from $Act^!(P, \Lambda)$ is henceforth a symbol $p!^a q$, which indicates that a message $a \in \Lambda$ is sent from process p to process q, and an action from $Act^?(P, \Lambda)$ is a symbol $q?^a p$, which represents the complementary action of receiving a so that we will write $(p!^a q, q?^a p) \in Com(P, \Lambda)$ or, more specifically, $(p!^a q, q?^a p) \in Com(P, \Lambda)_{(p,q)}$. As

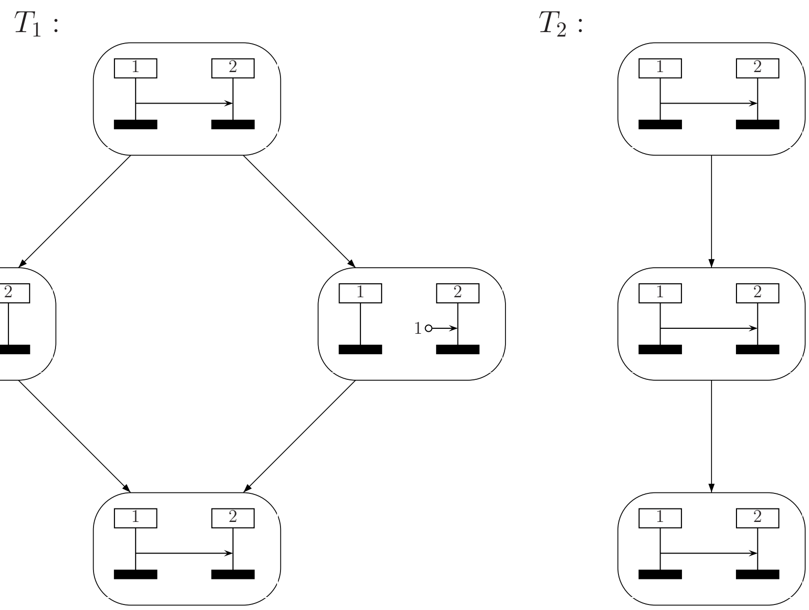


Figure 3.7: Two trace representations of one basic MSC

usual, the union of $Act^!(P,\Lambda)$ and $Act^?(P,\Lambda)$ is denoted by $Act(P,\Lambda)$. However, there is some scope for the canonical extension of the definition of an MSC and we consider the following definition to be a starting point. Let us define MSCs directly without going via partial MSCs first.

Definition 3.4.1 (extends Definition 3.1.3)
A *message sequence chart* over (P,Λ) is a structure $M = (E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda)$ from $\mathbb{DG}(Act(P,\Lambda), P_c)$ such that

- $\lhd_p$ is the covering relation of some total order $\leq_p$ on $\lambda^{-1}(Act(P,\Lambda)_p)$ (where $Act(P,\Lambda)_p$ is defined in the obvious manner),
- $\lhd_c \subseteq E \times E$ such that, for any $e, e' \in E$, $e \lhd_c e'$ iff $(\lambda(e), \lambda(e')) \in Com(P,\Lambda)$ and $|M \Downarrow e|_{\lambda(e)} = |M \Downarrow e'|_{\lambda(e')}$, and
- $|M|_{p!^a q} = |M|_{q?^a p}$ for any $(p,q) \in Ch$ and $a \in \Lambda$.

If we left this definition as it currently stands, we would allow non-FIFO behavior as depicted in Figure 3.8 on the next page where messages of different type overtake each other in channel $(1,2)$. Such a communication model is considered in [Mor02, Loh03, BM03]. Fortunately, though overtaking occurs, one can then still uniquely determine an MSC on the basis of a linearization or a collection of projections. This is no longer true if reversals between identical messages are allowed. In [BLN02], it is shown how to relate MSCs and their linearizations even in the that case, which, though it is called *degenerate* in [AEY00], is explicitly provided by the MSC standard.

For $(p,q) \in Ch$, let $Act^!(P,\Lambda)_{(p,q)}$ denote the set $\{p!^a q \mid a \in \Lambda\}$ and, accordingly, $Act^?(P,\Lambda)_{(p,q)}$ denote $\{q?^a p \mid a \in \Lambda\}$. If we still wish an MSC M to behave in a FIFO manner, we consequently require that, for any $e, e' \in E$, $e \lhd_{\mathrm{c}} e'$ iff, for some p and q, we have both $(\lambda(e), \lambda(e')) \in Com(P,\Lambda)_{(p,q)}$ and, moreover, $|M \Downarrow e|_{Act^!(P,\Lambda)_{(p,q)}} = |M \Downarrow e'|_{Act^?(P,\Lambda)_{(p,q)}}$. Then, for any $p, q \in P$, $e_1, e_2 \in E_p$, and $e_1', e_2' \in E_q$ with both $e_1 \lhd_{\mathrm{c}} e_1'$ and $e_2 \lhd_{\mathrm{c}} e_2'$, it holds $e_1 \leq_p e_2$ iff $e_1' \leq_q e_2'$.

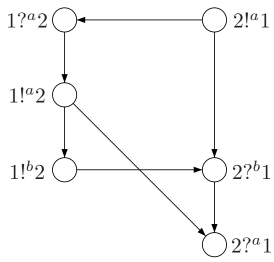


Figure 3.8: MSC with non-FIFO behavior

It remains to clarify what we understand by $\forall B$-/$\exists B$-boundedness for some $B \geq 1$, which just defers to the definition of a B-bounded word over $Act(P,\Lambda)$. So let us henceforth call a word $w \in \mathbb{W}(Act(P,\Lambda))$ $\forall B$-/$\exists B$-bounded if, for any prefix v of w and any $(p,q) \in Ch$,

$$\left(\sum_{a \in \Lambda} |v|_{p!^a q}\right) - \left(\sum_{a \in \Lambda} |v|_{q?^a p}\right) \leq B.$$

Whenever we refer to MSCs over (P,Λ), we just write $\mathbb{MSC}(P,\Lambda)$.

We would like to stress that, though we abstract from concrete messages in the following and almost exclusively consider MSCs over P, all the results of this thesis also hold in the scope of $\mathbb{MSC}$ over (P,Λ) and corresponding logics and automata, whose definition proceeds as anticipated.

3.5 High-Level Message Sequence Charts

Usually, a system designer wants a specification or an implementation to comprise many scenarios, which gives rise to an MSC language. In the following sections, we recall several rather algebraic characterizations of MSC languages, starting with *high-level MSCs*, a high-level construct, whose standard description of the norm Z.120 allows nondeterministic choice, concatenation, and iteration of MSCs for conveniently specifying possibly infinite sets of MSCs. Then, *regular languages* characterize sets of MSCs that, in some sense, are *realizable* and whose definition, though algebraically established, rather stems from a state-based implementation point of view. Other formalisms introduced in the rest of this chapter are *product languages*, which are closed under some independence operator an implementation might be based on, and EMSO-definable MSC languages, which will turn out to be the logical counterpart of our most general finite model of an implementation. Let us focus on high-level descriptions in this section, which are needed to combine MSCs and describe possibly infinitely many scenarios in a compact manner. In high-level MSCs, the central combination operator will be concatenation. Recall that we focus on *asynchronous* or *weak* concatenation, i.e., the only restriction on the order of events in the product of MSCs M and M' is that, for any process line, events of M precede events of M', while other events remain unordered unless they are otherwise causally ordered.

When introducing high-level descriptions, we come from their most general case. In our framework, partial MSCs correspond to what is commonly known as compositionality, which was introduced in [GMP01]. Compositional high-level constructions allow for the modeling of more complex systems than their simpler variant, particularly featuring MSC languages that are not finitely-generated. Basically, a *high-level compositional MSC* is a rational expression of $\mathrm{p}\mathbb{MSC}$ whose language, though consisting of partial MSCs, can be understood as a basic MSC language if previously unmatched events are combined along a FIFO architecture.

Definition 3.5.1 (High-Level Compositional MSC, cf. [GMP01])
A *high-level compositional MSC* (HcMSC) (over P) is a rational expression of $\mathrm{p}\mathbb{MSC}(P)$.

For readability, we omit henceforth any reference to P in this section. But note that the following definitions still depend on P.

In contrast to [GMP01] and rather following [Mor02], our approach to high-level (compositional) MSCs is algebraically motivated and, based on the idea of prime MSCs, allows to relate the corresponding MSC languages to trace theory. Moreover, we follow the approach adopted in [AEY01, HMKT00b, Mad01, MKS00], where high-level MSCs are flattened into *message sequence graphs*.

The *MSC language* of an HcMSC $\mathcal{H}$, which we denote by $\mathfrak{L}(\mathcal{H})$, is defined to be the set $\{M' \in \mathbb{MSC} \mid M'$ is represented by some $M \in L(\mathcal{H})\}$ (recall that, in contrast, $L(\mathcal{H})$ is primarily a set of partial MSCs). If we say that HcMSCs $\mathcal{H}$ and $\mathcal{H}'$ are *equivalent*, we actually mean $\mathfrak{L}(\mathcal{H}) = \mathfrak{L}(\mathcal{H}')$ in the following.

An HcMSC might be seen as a structure $\mathcal{H} = (V, R, v^{in}, V^f, \mu)$ where V is the nonempty finite set of *nodes*, $R \subseteq V \times V$ is the set of *transitions*, $v^{in} \in V$ is the *initial node*, $V^f \subseteq V$ is the set of *final nodes*, and μ is a mapping $V \to \mathrm{p}\mathbb{MSC}$. An *execution* of $\mathcal{H}$ is henceforth a sequence $v_0 \ldots v_n \in V^+$ of nodes such that $v_0 = v^{in}$ and $(v_i, v_{i+1}) \in R$ for any $i \in \{0, \ldots, n-1\}$. It is called *accepting* if, moreover, $v_n \in V^f$. An execution $\rho = v_0 \ldots v_n$ of $\mathcal{H}$ gives rise to the partial MSC $M(\rho) := \mu(v_0) \cdot \ldots \cdot \mu(v_n)$. Then, the language $L(\mathcal{H})$ can be defined to be $\{M(\rho) \mid \rho$ is an accepting execution of $\mathcal{H}\}$. Observe that this view of an HcMSC is equivalent to the one proposed before. Some HcMSCs are depicted in Figure 3.11 on page 54, while Example 3.5.5 on the next page will make use of their representation as rational expressions.

Definition 3.5.2 (Safe and Left-Closed HcMSC)
An HcMSC $\mathcal{H}$ is called

- *safe* if any $M \in L(\mathcal{H})$ represents some basic MSC and
- *left-closed* if, for any execution ρ of $\mathcal{H}$ (accordingly defined for rational expressions), $M(\rho)$ represents some left-closed partial MSC.

It is easy to show that any left-closed HcMSC is equivalent to some safe and left-closed HcMSC. Note that, in [GMP01], left-closed HcMSCs are called *realizable*.

Many interesting properties are undecidable for sets of MSCs formalized by those high-level descriptions. Inspired by the notion of a star-connected rational expression in the theory of Mazurkiewicz traces, the more restrictive but useful notion of *global cooperation* in high-level MSCs was introduced independently in [Mor02] and [GMSZ02] and extended to HcMSCs in [GKM04].

Definition 3.5.3 (Globally-Cooperative HcMSC)
An HcMSC is called *globally-cooperative* (gc-HcMSC) if iteration occurs over sets of connected partial MSCs only.

Note that, actually, [GKM04] makes use of a different notion of global cooperativity, which is more general in the context of safe HcMSCs and requires that iteration occurs over sets of partial MSCs with connected communication graph. However, in the special case of HMSCs, which are defined as follows, their condition captures exactly the same expressions as our notion.

Definition 3.5.4 (High-Level Message Sequence Chart)
A *high-level message sequence chart* (HMSC) is an HcMSC that is built from MSCs only, i.e., it is a rational expression of $\mathbb{MSC}$.

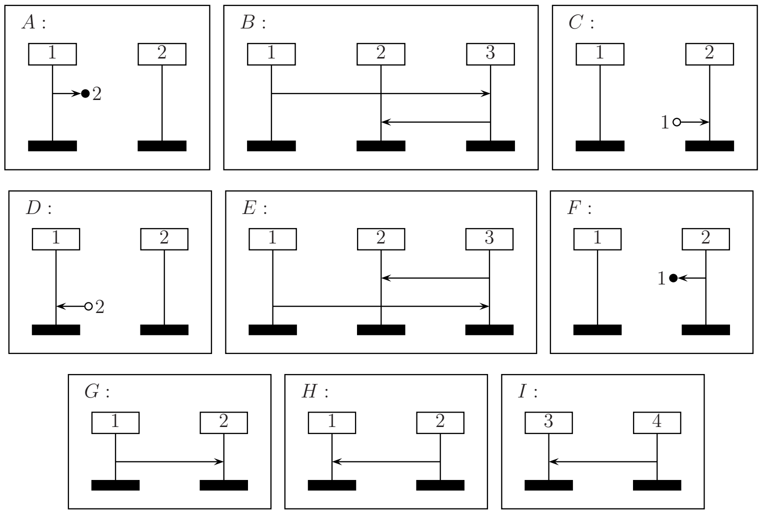


Figure 3.9: The building blocks of HcMSCs

Thus, an HMSC is trivially safe and left-closed. Moreover, the MSC language $\mathcal{L}(\mathcal{H}) = L(\mathcal{H})$ of some HMSC $\mathcal{H}$ is finitely generated. Thus, HMSCs are in general not capable of defining the MSC language of the HcMSC $\mathcal{H}_\mathrm{a}$ depicted in Figure 3.11 on page 54, which defines MSCs in the style of M_a from Figure 3.10.

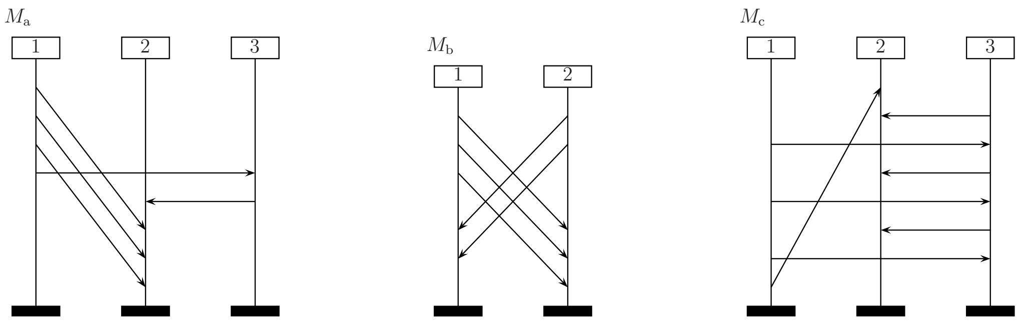


Figure 3.10: The basic MSC languages of some gc-HcMSCs

Example 3.5.5 Consider the partial MSCs from Figure 3.9 and the following HcMSCs, whose graphs (which are not unique, of course) are illustrated in Figure 3.11 on page 54.

a) The gc-HcMSC $\mathcal{H}_\mathrm{a} = A^+ \cdot B \cdot C^+$, whose basic MSC language features, among others, the basic MSC M_a from Figure 3.10, is neither safe nor left-closed.

b) The gc-HcMSC $\mathcal{H}_\mathrm{b} = A^+ \cdot F^+ \cdot D^+ \cdot C^+$ is also neither safe nor left-closed.

It defines the basic MSC language whose basic MSCs look like M_{b} from Figure 3.10.

c) $\mathcal{H}_{\mathrm{c}} = C \cdot E^{+} \cdot A$ is a safe, though not left-closed, gc-HcMSC, which defines the set of basic MSCs in the style of M_{c} from Figure 3.10. Note that there is no left-closed HcMSC equivalent to $\mathcal{H}_{\mathrm{c}}$.

d) $\mathcal{H}_{\mathrm{d}} = A \cdot B^{+} \cdot C$ is a gc-HcMSC, which is both safe and left-closed.

e) Finally, $\mathcal{H}_{\mathrm{e}} = G \cdot ((H \cdot G) + (I \cdot G))^{*}$ is an HMSC, as it is composed of MSCs only. However, $\mathcal{H}_{\mathrm{e}}$ is *not* globally-cooperative and has no equivalent gc-HcMSC counterpart either.

Note that $\mathcal{H}_{\mathrm{a}}$ and $\mathcal{H}_{\mathrm{b}}$ are both *not* equivalent to some safe gc-HcMSC. But even if $\mathcal{H}_{\mathrm{a}}$ is not safe, it is a natural specification, generating a quite simple MSC language. Having in mind $\mathcal{H}_{\mathrm{a}}$, the system designer is usually unconcerned about the channel architecture and does not care if, in every run of its specification, the number of sends equals the number of receives. In fact, this is the job of the channel architecture of an implementation, which justifies that we consider HcMSCs in their most general form.

Let us define the language classes associated with HcMSCs and their restrictions. We set

- $\mathcal{H}\mathrm{c}\mathcal{MSC} := \{L \subseteq \mathbb{MSC} \mid L = \mathfrak{L}(\mathcal{H})$ for some HcMSC $\mathcal{H}\}$,
- gc-$\mathcal{H}\mathrm{c}\mathcal{MSC} := \{L \subseteq \mathbb{MSC} \mid L = \mathfrak{L}(\mathcal{H})$ for some gc-HcMSC $\mathcal{H}\}$,
- safe-gc-$\mathcal{H}\mathrm{c}\mathcal{MSC} := \{L \subseteq \mathbb{MSC} \mid L = \mathfrak{L}(\mathcal{H})$ for some safe gc-HcMSC $\mathcal{H}\}$,
- left-closed-gc-$\mathcal{H}\mathrm{c}\mathcal{MSC} := \{L \subseteq \mathbb{MSC} \mid L = \mathfrak{L}(\mathcal{H})$ for some left-closed gc-HcMSC $\mathcal{H}\}$,
- $\mathcal{HMSC} := \{L \subseteq \mathbb{MSC} \mid L = \mathfrak{L}(\mathcal{H})$ for some HMSC $\mathcal{H}\}$, and
- gc-$\mathcal{HMSC} := \{L \subseteq \mathbb{MSC} \mid L = \mathfrak{L}(\mathcal{H})$ for some gc-HMSC $\mathcal{H}\}$.

Note that we have $\mathcal{HMSC} = \mathcal{RAT}_{\mathbb{MSC}}$, as any HMSC is a rational expression of $\mathbb{MSC}$.

Proposition 3.5.6 The classes of HcMSCs form the hierarchy depicted in Figure 3.12 on page 55. The hierarchy is strict.

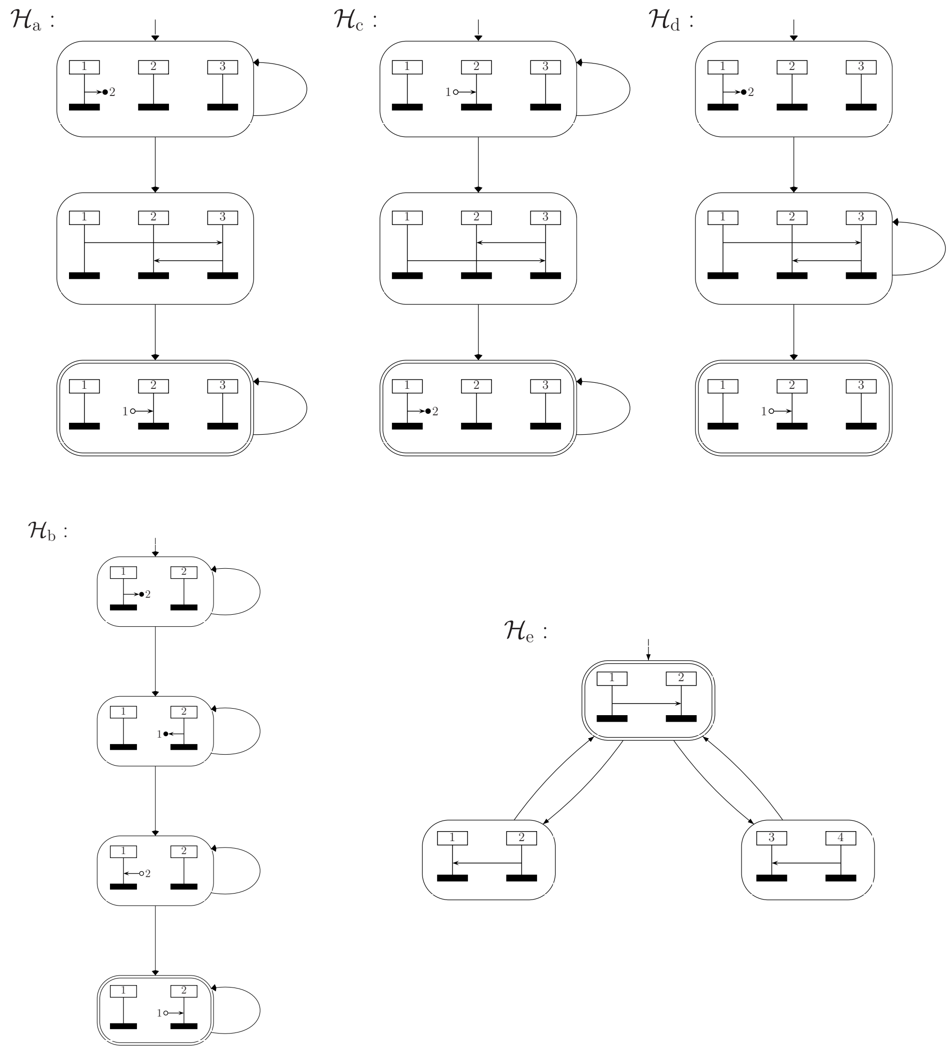


Figure 3.11: HcMSCs as graphs

Proof Inclusions follow directly from the definitions. Moreover, strictness is witnessed by the HcMSCs from Example 3.5.5, which are assigned in Figure 3.12 to a class they belong to so that there is no equivalent counterpart in a lower class, respectively. □

While we first proposed HcMSCs as the most general case of our specification language, HMSCs were actually the starting point of everything [MR97]. The study of compositionality was then initiated in [GMP01] by Gunter et al. The notion of global cooperation was independently introduced by Morin in [Mor02] and Genest et al. in [GMSZ02], who also proposed the notion of *local cooperation* and showed that any *local-choice HMSC* is implementable without deadlocks. Since

$$
\begin{array}{cc}
\mathcal{H}\mathrm{c}\mathcal{MSC} & (\mathcal{H}_{\mathrm{e}}) \\
\uparrow & \\
\text{gc-}\mathcal{H}\mathrm{c}\mathcal{MSC} & (\mathcal{H}_{\mathrm{a}},\, \mathcal{H}_{\mathrm{b}}) \\
\uparrow & \\
\text{safe-gc-}\mathcal{H}\mathrm{c}\mathcal{MSC} & (\mathcal{H}_{\mathrm{c}}) \\
\uparrow & \\
\text{left-closed-gc-}\mathcal{H}\mathrm{c}\mathcal{MSC} & (\mathcal{H}_{\mathrm{d}}) \\
\uparrow & \\
\text{gc-}\mathcal{HMSC} &
\end{array}
$$

Figure 3.12: The hierarchy of HcMSCs

then, several extensions have been considered to facilitate and enrich specification of communicating systems. For instance, [LMM02] introduces a formalism that allows for generating processes during system execution, while we assume the number of processes to be fixed. In [MKT03] and [BM04], *netcharts* are proposed and studied, which combine HMSCs with Petri nets to generate an executable behavior. Moreover, *live sequence charts* allow to distinguish between mandatory and provisional system behavior, which is reflected in *universal* and *existential* charts, respectively [DH01]. Another important area, which, however, is not considered in this thesis, is verification of specifications against formalisms such as temporal logics [MR00, BL01, MR04, GMMP04], monadic second-order logics [Mad01, MM01, LMM02], and *template* MSCs [MPS98, GMMP04].

3.6 Regular MSC Languages

There have been several proposals for the *right* notion of regularity for MSC languages. In their seminal work [HMKT00b, HMK$^+$05], Henriksen et al. consider an MSC language to be regular if its set of linearizations forms a regular word language. For example, the MSC language $\{G\}^*$ (where G is taken from Figure 3.9 on page 52, which allows to concatenate G arbitrarily often, is not bounded and hence cannot be regular. It induces the set of MSCs that send arbitrarily many messages from process 1 to process 2. The set of corresponding linearizations gives rise to a set of words that, in particular, show the same number of send and receive events. This language is not recognizable in the free word monoid, as, intuitively, we would need an unbounded counter for the number of messages not being received. In contrast, the language $\{G \cdot H\}^*$ is regular, as its word language can be easily realized by a finite automaton. In simple words, regularity aims at finiteness of the underlying *global* system, which incorporates the state of a communication channel.

Definition 3.6.1 (Regular MSC Language [HMKT00b])
A set $L \subseteq \mathbb{MSC}$ is called a *regular MSC language* (over P) if $Lin(L)$ is a regular word language over Act, i.e., $Lin(L) \in \mathcal{REC}_{\mathbb{W}(Act)}$.

The class of regular MSC languages is denoted by $\mathcal{R}_{\mathbb{MSC}}$.

Lemma 3.6.2 ([HMKT00b]) Any regular MSC language is $\forall$-bounded.

As mentioned above, $\{G\}^*$ with G again taken from Figure 3.9 is not regular. However, it is $\exists$1-bounded and there is a simple gc-HMSC defining this language. Moreover, as we will see in the next section, there is a simple finite message-passing automaton accepting $\{G\}^*$. Thus, we are looking for another, extended notion of *regularity*. So let us in the next section examine monadic second-order logics whose formulas are interpreted over MSCs.

3.7 (E)MSO-definable MSC Languages

Recall that an MSC is modeled as a graph, which corresponds to the view taken in [Mad01] and also adopted by [LL95, BAL97] where (the graph of) an MSC is called a *message flow graph*. However, while most theorems (among them Theorem 5.2.6) hold independently of the modeling, the way to define an MSC immediately affects the syntax and expressivity of (fragments of) the corresponding monadic second-order logic. As $\mathbb{MSC}(P) \subseteq \mathbb{DG}(Act, P_c)$, the monadic second-order formulas that can be applied to MSCs, are those from $\mathrm{MSO}(Act, P_c)$. Recall that the corresponding atomic entities are

$$\lambda(x) = \sigma \text{ (for } \sigma \in Act) \qquad x \lhd_p y \text{ (for } p \in P) \qquad x \lhd_c y \qquad x \in X \qquad x = y$$

(where $x, y \in \mathrm{Var}$ and $X \in \mathrm{VAR}$). The definition of their semantics arises from the general case of graphs.
The formula $P(x) = p$ will serve as a shorthand for $\bigvee_{\sigma \in Act_p} \lambda(x) = \sigma$.

Example 3.7.1 Let L be the set of MSCs whose linearizations are of the form $(1!2)^n(1!3)(3?1)(3!2)(2?3)(2?1)^n$, $n \in \mathbb{N}$. Recall that $L \in$ gc-$\mathcal{H}$c$\mathcal{MSC}$ and that an example MSC from L is provided by M_a from Figure 3.10 on page 52. In fact, L is $\mathrm{MSO}_{\mathbb{MSC}}$-definable. Let φ be the conjunction of the formulas φ_1, φ_2, and φ_3, which describe the behavior of processes 1, 2, and 3, respectively, and are given as follows:

$$\begin{aligned}
\varphi_1 \quad = \quad \exists x(& \quad \lambda(x) = 1!3 \\
& \wedge \ \forall y(P(y) = 1 \wedge y <_1 x \rightarrow \lambda(y) = 1!2) \\
& \wedge \ \forall y(P(y) = 1 \wedge x \leq_1 y \rightarrow y = x))
\end{aligned}$$

$$\begin{aligned}
\varphi_2 \quad = \quad \exists x(& \quad \lambda(x) = 2?3 \\
& \wedge \ \forall y(P(y) = 2 \wedge x <_2 y \rightarrow \lambda(y) = 2?1) \\
& \wedge \ \forall y(P(y) = 2 \wedge y \leq_2 x \rightarrow y = x))
\end{aligned}$$

$$\begin{aligned}
\varphi_3 \quad = \quad \exists x(& \quad \lambda(x) = 3?1 \\
& \wedge \ \forall y(y \leq_3 x \rightarrow y = x))
\end{aligned}$$

It holds $L_{\mathbb{MSC}}(\varphi) = L$. Moreover, φ can be equivalently written as both an EMSO-formula and, even more obviously, an FO[$\leq$]-formula. Note that, as L is a set of total orders that gives rise to a non-regular word language over Act, L is not $\mathrm{MSO}(Act, -)_{\mathbb{W}(Act)}$-definable, when formulas are interpreted over (arbitrary) words. However, as we interpret φ over MSCs, only those words have to be considered that are linearizations of MSCs. Accordingly, φ rather defines total orders *generated* by the rational expression $(1!2)^*(1!3)(3?1)(3!2)(2?3)(2?1)^*$ while restricting to MSCs only rules out graphs that are not valid MSCs.

Recall that the class of $\mathrm{MSO}_{\mathbb{MSC}}$-definable MSC languages is denoted by $\mathcal{MSO}_{\mathbb{MSC}}$ and the one of $\mathrm{EMSO}_{\mathbb{MSC}}$-definable MSC languages by $\mathcal{EMSO}_{\mathbb{MSC}}$.

3.8 Product MSC Languages

Languages defined by finite transition systems working in parallel are known as *product languages* and were initially studied by Thiagarajan in [Thi95] in the domain of Mazurkiewicz traces where distributed components communicate executing actions simultaneously rather than sending messages (cf. Section 2.4). Taking up the idea of product behavior, [AEY00] considers MSC languages that are closed under *inference*, which can be described by the setting depicted in Figure 3.9 on page 52. Attempting to realize the MSC language $\{G, I\}$, one might argue that the behavior of $G \cdot I$ is a feasible one, too. As processes 1 and 2 do not get in touch with processes 3 and 4, it is not clear to a single process whether to realize the behavior of G or that of I so that, finally, $G \cdot I$ might be *inferred* from $\{G, I\}$. We call a set of MSCs that is closed under such an inference a *weak product MSC language*. Note that, in [AEY00], no finiteness condition was studied so that, in principle, it is possible to *realize* $\{G^{n^2} \mid n \in \mathbb{N}_{\geq 1}\}$. Summarizing, we may say that product behavior respects *independence.*

Let us be more precise and, given $L \subseteq \mathbb{MSC}$ and $M \in \mathbb{MSC}$, write $L \vdash_P M$ if the following holds:

$$\forall p \in P: \ \exists M' \in L: \ M' \upharpoonright p = M \upharpoonright p$$

Definition 3.8.1 ((Weak) Product MSC Language, cf. [AEY00])
A set $L \subseteq \mathbb{MSC}$ is called a *weak product MSC language* (over P) if, for any $M \in \mathbb{MSC}$, $L \vdash_P M$ implies $M \in L$. The finite union of weak product MSC languages is called a *product MSC language* (over P).

Adopting the notation we introduced for traces, we denote by $\mathcal{P}^0_{\mathbb{MSC}}$ the class of weak product MSC languages and by $\mathcal{P}_{\mathbb{MSC}}$ the class of product MSC languages. In other words, an MSC language L is a weak product MSC language if every MSC that agrees on each process line with some MSC from L is contained in L, too. Getting back to Figure 3.9, $G \cdot I$ agrees with G on the first two process lines and with I on the remaining two. Thus, $G \cdot I$ belongs to any weak product language containing both G and I. As global knowledge of an underlying system, one often allows several global initial or final states. This is the reason for considering finite unions of weak product languages. For example, $\{G, I\}$ is a product MSC language, while $\{G \cdot I\}^*$ is not. Let us again bring together the concepts of product behavior and regularity.

Definition 3.8.2 (Regular Product MSC Language)
We call $\mathcal{R}_{\mathbb{MSC}} \cap \mathcal{P}^0_{\mathbb{MSC}}$ the class of *weak regular product MSC languages* (over P) and denote it by $\mathcal{RP}^0_{\mathbb{MSC}}$. Moreover, an MSC language L is said to be a *regular product MSC language* (over P), written $L \in \mathcal{RP}_{\mathbb{MSC}}$, if it is the finite union of sets from $\mathcal{RP}^0_{\mathbb{MSC}}$.

Let us now extend our study towards product languages in combination with EMSO-definable languages. As the class of EMSO-definable languages will turn out to capture exactly the class of languages implementable in terms of a finite message-passing automaton, we rather concentrate on EMSO-definable languages than on MSO-definable ones.

Definition 3.8.3 (EMSO-definable Product MSC Language)
We call $\mathcal{EMSO}_{\mathbb{MSC}} \cap \mathcal{P}^0_{\mathbb{MSC}}$ the class of *weak EMSO-definable product MSC languages* and denote it by $\mathcal{EP}^0_{\mathbb{MSC}}$. An MSC language L is called an *EMSO-definable product MSC language* ($L \in \mathcal{EP}_{\mathbb{MSC}}$) if it is the finite union of sets from $\mathcal{EP}^0_{\mathbb{MSC}}$.

If it is clear from the context that we talk about MSCs, we omit the reference to $\mathbb{MSC}$, speak of, for example, MSO- and EMSO-definability, and simply write $\mathcal{R}$, $\mathcal{P}$, and $\mathcal{MSO}$.

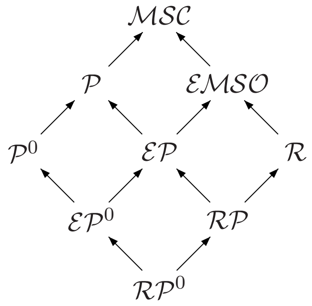
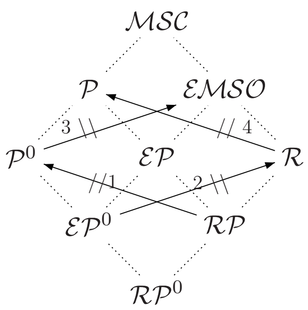


Figure 3.13: The hierarchy of product MSC languages

Theorem 3.8.4 The classes of languages proposed so far (apart from $\mathcal{H}c\mathcal{MSC}$, which is reconsidered in Chapter 5 in more detail) draw the picture shown in Figure 3.13. The hierarchy is strict.

Proof We will prove $\mathcal{R} \subseteq \mathcal{EMSO}$ in Chapter 4. The other inclusions are straightforward. It remains to show strictness and incomparability. Consider the MSCs G and I from Figure 3.9 on page 52. For a (crossed) arrow from a class of MSC languages $\mathfrak{C}_1$ to a class $\mathfrak{C}_2$ in Figure 3.14, the tabular aside specifies an MSC language L with $L \in \mathfrak{C}_1$ and $L \notin \mathfrak{C}_2$. $\square$



1	$\{G, I\}$
2	$\{G\}^*$
3	$L(f_k)$ with $k \geq 2$ (cf. Theorem 5.2.1)
4	$L_{\widehat{N}+1}$ with $\widehat{N} \geq 1$ (cf. Lemma 4.5.2)

Figure 3.14: Strictness and incomparability in the hierarchy

Chapter 4

Message-Passing Automata

In this chapter, we introduce and study *message-passing automata* (MPAs), a model of computation rather than a specification language, which is close to a real-life implementation of a communicating system.

4.1 Message-Passing Automata

MPAs can be considered to be the most common computation model for MSCs. However, slightly different but related models have been investigated in [BZ83, MKRS98], for example. An MPA is a collection of state machines that share one global initial state and several global final states. The machines are connected pairwise with a priori unbounded reliable FIFO buffers. The transitions of each component are labeled with send or receive actions. Hereby, a send action $p!q$ puts a message at the end of the channel from p to q. A receive action can be taken provided the requested message is found in the channel. To extend the expressive power, MPAs can send certain *synchronization messages*. Let us be more precise:

Definition 4.1.1 (Message-Passing Automaton)
A *message-passing automaton* (over P) is a structure $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$ such that

- $\mathcal{D}$ is a nonempty finite set of *synchronization messages* (or *data*),
- for each $p \in P$, $\mathcal{A}_p$ is a pair (S_p, Δ_p) where
  - S_p is a nonempty set of (p-)*local states* and
  - $\Delta_p \subseteq S_p \times Act_p \times \mathcal{D} \times S_p$ is the set of (p-)*local transitions*,

- $\overline{s}^{in} \in \prod_{p \in P} S_p$ is the *global initial state*, and
- $F \subseteq \prod_{p \in P} S_p$ is the finite set of *global final states*.

By $S_{\mathcal{A}}$, we denote the set $\prod_{p \in P} S_p$ of *global states* of $\mathcal{A}$. For $\overline{s} = (s_p)_{p \in P} \in S_{\mathcal{A}}$, $\overline{s}[p]$ will henceforth refer to s_p.

An MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, is called

- an N-MPA, $N \geq 1$, if $|\mathcal{D}| = N$,
- finite if, for each $p \in P$, S_p is finite,
- *locally-accepting* if, for any $p \in P$, there is a set $F_p \subseteq S_p$ such that $F = \prod_{p \in P} F_p$, and
- *deterministic* if, for any $p \in P$, Δ_p satisfies the following conditions:
  - If $(s, p!q, m_1, s_1) \in \Delta_p$ and $(s, p!q, m_2, s_2) \in \Delta_p$, then $m_1 = m_2$ and $s_1 = s_2$.
  - If $(s, p?q, m, s_1) \in \Delta_p$ and $(s, p?q, m, s_2) \in \Delta_p$, then $s_1 = s_2$.

The class of MPAs over P is denoted by $\mathrm{MPA}(P)$[1], the class of finite MPAs by $\mathrm{MPA}^f(P)$. However, as the underlying set of processes will be clear from the context, we henceforth omit any reference to P and just write MPA and, respectively MPA^f. For a set $\mathfrak{C}$ of MPAs, we denote by N-$\mathfrak{C}$, $\mathfrak{C}_\ell$, and det-$\mathfrak{C}$ the classes of N-, locally-accepting, and deterministic MPAs $\mathcal{A}$ with $\mathcal{A} \in \mathfrak{C}$, respectively. The intuition behind local acceptance will be that recognition by the whole system defers to acceptance by any single component. A locally-accepting finite 2-MPA with set of synchronization messages $\{\circ, \bullet\}$, which is not deterministic, is illustrated in Figure 4.1 on the next page. (Whenever we deal with local acceptance, we might depict a local final state by a second circle.) Note that its MSC language cannot be recognized by some MPA with only one synchronization message, even if we allow infinite local state spaces. Nevertheless, it can be recognized by some deterministic finite MPA. (To verify this is left to the reader as an exercise. Basically, the second component $\mathcal{A}_2$ has to be modified accordingly.) However, we should first define the behavior of MPAs. In doing so, we adhere to the style of [Kus03]. In particular, an automaton will run on MSCs rather than linearizations of MSCs, allowing for its distributed behavior. Let $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, be an MPA and $M = (E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda) \in \mathbb{MSC}$ be an MSC. For

[1]Note that $\mathrm{MPA}(P)$ does not impose any restriction on the state space, which can therefore produce non-recursive behavior.

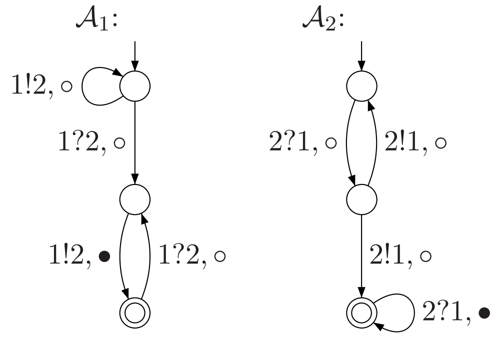


Figure 4.1: A message-passing automaton

a function $r : E \to \bigcup_{p \in P} S_p$, we define $r^- : E \to \bigcup_{p \in P} S_p$ to map an event $e \in E$ onto $\overline{s}^{in}[P(e)]$ if e is minimal in $(E_{P(e)}, \leq_{P(e)})$ and, otherwise, onto $r(e')$ where $e' \in E_{P(e)}$ is the unique event with $e' \lessdot_{P(e)} e$. A *run* of $\mathcal{A}$ on M is a pair (r, m) of mappings $r : E \to \bigcup_{p \in P} S_p$ with $r(e) \in S_{P(e)}$ for each $e \in E$ and $m : \lhd_c \to \mathcal{D}$ such that, for any $e, e' \in E$, $e \lhd_c e'$ implies

– $(r^-(e), \lambda(e), m((e, e')), r(e)) \in \Delta_{P(e)}$ and

– $(r^-(e'), \lambda(e'), m((e, e')), r(e')) \in \Delta_{P(e')}$.

For $p \in P$, let f_p denote $\overline{s}^{in}[p]$ if E_p is empty. Otherwise, let f_p denote the p-local state $r(\mathit{last}(M \upharpoonright p))$. We call (r, m) *accepting* if $(f_p)_{p \in P} \in F$. By $L(\mathcal{A}) := \{M \in \mathbb{MSC} \mid$ there is an accepting run of $\mathcal{A}$ on $M\}$, let us denote the *language* of $\mathcal{A}$. Moreover, we may canonically define $\mathbf{1}_{\mathbb{MSC}}$ to be a member of $L(\mathcal{A})$ if (and only if) $\overline{s}^{in} \in F$. Given a class $\mathfrak{C}$ of MPAs, we furthermore set $\mathcal{L}(\mathfrak{C})$ to be $\{L \subseteq \mathbb{MSC} \mid$ there is $\mathcal{A} \in \mathfrak{C}$ such that $L = L(\mathcal{A})\}$, which is the class of languages of $\mathfrak{C}$. Moreover, we consider $\mathcal{L}(\mathrm{MPA}^f)$ to be some kind of standard class, which is identified by $\mathcal{MPA} := \mathcal{L}(\mathrm{MPA}^f)$. We also say that the languages from $\mathcal{MPA}$ are the *implementable* ones. This nomenclature is arbitrary and rather geared to the literature, where the term realizability usually refers to locally-accepting 1-MPAs.

Note that, for any deterministic finite MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, and any MSC M, there is at most one run of $\mathcal{A}$ on M. However, $\mathcal{A}$ can be extended towards a deterministic finite MPA $\mathcal{A}' = ((\mathcal{A}'_p)_{p \in P}, \mathcal{D}', \overline{s}'_0, F')$, $\mathcal{A}'_p = (S'_p, \Delta'_p)$, such that $L(\mathcal{A}') = L(\mathcal{A})$ and, for any MSC M, there is *exactly* one run of $\mathcal{A}'$ on M. To this aim, we extend the set of synchronization messages $\mathcal{D}$ by a message *fail*, i.e., $\mathcal{D}' = \mathcal{D} \uplus \{\mathit{fail}\}$ and, for each $p \in P$, enrich S_p by a sink state sink_p, i.e., $S'_p = S_p \uplus \{\mathit{sink}_p\}$. For any $p \in P$, Δ'_p already contains all the transitions from Δ_p. Moreover, transitions are added to Δ'_p according to the following procedure: for any $s \in S_p$ and $q \in P \setminus \{p\}$,

- if there is no $m \in \mathcal{D}$ and $s' \in S_p$ such that $(s, p!q, m, s') \in \Delta_p$, add a transition $(s, p!q, \mathit{fail}, \mathit{sink}_p)$ and
- add a transition $(\mathit{sink}_p, p!q, \mathit{fail}, \mathit{sink}_p)$.

For any $s \in S_p$, $q \in P \setminus \{p\}$, and $m \in \mathcal{D}$,

- if there is no $s' \in S_p$ such that $(s, p?q, m, s') \in \Delta_p$, add a transition $(s, p?q, m, \mathit{sink}_p)$,
- add a transition $(s, p?q, \mathit{fail}, \mathit{sink}_p)$, and
- add a transition $(\mathit{sink}_p, p?q, m', \mathit{sink}_p)$ for any $m' \in \mathcal{D}'$.

Finally, $\overline{s}^{in}$ and F are adopted from $\mathcal{A}$, i.e., $\overline{s}'_0 = \overline{s}^{in}$ and $F' = F$. In other words, local states from S_p that lack an outgoing send transition labeled $p!q$ for some q, are enabled to send at least a message *fail* to q, though this is doomed to failure. Moreover, a missing receipt of a message m is made possible, even if it ends in a state sink_p, which does not contribute to an accepting run either. Thus, the above transformation does not affect the recognized language but enables the resulting automaton to execute a run on any given MSC. Those automata will be used to show in Chapter 5 that deterministic finite MPAs are strictly weaker than their nondeterministic counterpart.

Consider two variants of MPAs. The first allows for accepting extended MSCs, say from $\mathbb{MSC}^Q$ for some alphabet Q. Accordingly, for $p \in P$, the p-local transition relation of an MPA is henceforth a subset of $S_p \times (\mathit{Act}_p \times Q) \times \mathcal{D} \times S_p$. However, the type of an action (σ, q) still depends only on σ so that, in particular, a run may allow communicating events to have different additional labelings. Such an automaton will be used in Chapter 5 to characterize the language of some EMSO(Act, P_c)-formula $\varphi(Y_1, \ldots, Y_n)$, which, as mentioned in Chapter 2, defines a subset of $\mathbb{MSC}^{\{0,1\}^n}$. A second variant of MPAs allows to specify different starting points of a run instead of one single global initial state. So let an *extended* MPA be an MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, S^{in}, F)$ where, though, $S^{in} \subseteq \prod_{p \in P} S_p$ is a nonempty finite set of *global initial states*. An (accepting) run of $\mathcal{A}$ and the language of $\mathcal{A}$ are defined analogously to the MPA case: an (accepting) run of $\mathcal{A}$ is an (accepting, respectively) run of one of the MPAs $((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}_0, F)$ with $\overline{s}_0 \in S^{in}$. As it will turn out, several global initial states do not extend expressiveness.

In [HMKT00b, Mor02, Gen04], a run of an MPA is defined on linearizations of MSCs rather than on MSCs, which reflects an operational behavior at the expense that several execution sequences might stand for one and the same run. Usually, such a view relies on the *global transition relation* of $\mathcal{A}$, which, in turn, defers to the notion of a *configuration.* Let us be more precise and consider an MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$. The set of *configurations* of $\mathcal{A}$, denoted by $Conf_{\mathcal{A}}$, is the cartesian product $S_{\mathcal{A}} \times \mathcal{C}_{\mathcal{A}}$ where $\mathcal{C}_{\mathcal{A}} := \{\chi \mid \chi : Ch \to \mathcal{D}^*\}$ is the set of possible *channel contents* of $\mathcal{A}$. Now, the *global transition relation* of $\mathcal{A}$, $\Longrightarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times Act \times \mathcal{D} \times Conf_{\mathcal{A}}$, is defined as follows:

– $((\overline{s}, \chi), p!q, m, (\overline{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}$ if

- $(\overline{s}[p], p!q, m, \overline{s}'[p]) \in \Delta_p$,
- $\chi' = \chi[(p,q)/m \cdot \chi((p,q))]$ (i.e., χ' maps (p,q) to $m \cdot \chi((p,q))$ and, otherwise, coincides with χ), and
- for all $r \in P \setminus \{p\}$, $\overline{s}[r] = \overline{s}'[r]$.

– $((\overline{s}, \chi), p?q, m, (\overline{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}$ if there is a word $w \in \mathcal{D}^*$ such that

- $(\overline{s}[p], p?q, m, \overline{s}'[p]) \in \Delta_p$,
- $\chi((q,p)) = w \cdot m$,
- $\chi' = \chi[(q,p)/w]$, and
- for all $r \in P \setminus \{p\}$, $\overline{s}[r] = \overline{s}'[r]$.

Let $\chi_\varepsilon : Ch \to \mathcal{D}^*$ map each channel onto the empty word. When we set $(\overline{s}^{in}, \chi_\varepsilon)$ to be the *initial configuration* and $F \times \{\chi_\varepsilon\}$ to be the set of *final configurations*, $\mathcal{A}$ defines in the canonical way a word language $L_w(\mathcal{A}) \subseteq \mathbb{W}(Act)$. As one can verify, it holds $L_w(\mathcal{A}) = Lin(L(\mathcal{A}))$. In particular, $L_w(\mathcal{A})$ uniquely determines an MSC language. For $(\overline{s}, \chi), (\overline{s}', \chi') \in Conf_{\mathcal{A}}$, we write $(\overline{s}, \chi) \Longrightarrow_{\mathcal{A}} (\overline{s}', \chi')$ if $((\overline{s}, \chi), \sigma, m, (\overline{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}$ for some $\sigma \in Act$ and $m \in \mathcal{D}$. We call a configuration $(\overline{s}, \chi) \in Conf_{\mathcal{A}}$ *reachable* in $\mathcal{A}$ if $(\overline{s}^{in}, \chi_\varepsilon) \Longrightarrow_{\mathcal{A}}^* (\overline{s}, \chi)$.

For $B \geq 1$, an MPA $\mathcal{A}$ is called universally-*B*-*bounded* ($\forall B$-bounded) if $L(\mathcal{A})$ is $\forall B$-bounded.[2] Furthermore, $\mathcal{A}$ is called *universally-bounded* ($\forall$-bounded) if it is $\forall B$-bounded for some $B \geq 1$. Note that $\mathcal{A}$ is $\forall$-bounded if only a finite number of configurations is reachable in $\mathcal{A}$. Given a class $\mathfrak{C}$ of MPAs, let $\forall\mathfrak{C}$ denote the set of $\forall$-bounded MPAs $\mathcal{A}$ with $\mathcal{A} \in \mathfrak{C}$. The same principle as for universal boundedness applies to the existential one. In this sense, let $\exists\mathrm{MPA}^f$ denote the class of $\exists$-bounded finite MPAs. For example, the MPA from Figure 4.1 on page 63 is neither $\forall$-bounded nor $\exists$-bounded. Note that our definition of boundedness for MPAs coincides with the one used in [Kus03]. According to Henriksen et al., who

[2] Note that this is a semantic characterization, as it depends on the language of $\mathcal{A}$.

use a slightly different notion of bounded MPAs, we call an MPA $\mathcal{A}$ *strongly-$\forall B$-bounded* for some $B \geq 1$ if, for any $(p,q) \in Ch$ and any configuration $(\overline{s}, \chi)$ that is reachable in $\mathcal{A}$, $|\chi((p,q))| \leq B$. An MPA $\mathcal{A}$ is called *strongly-$\forall$-bounded* if it is strongly-$\forall B$-bounded for some $B \geq 1$. Given a class $\mathfrak{C}$ of MPAs, let $\forall_! \mathfrak{C}$ denote the set of strongly-$\forall$-bounded MPAs $\mathcal{A}$ with $\mathcal{A} \in \mathfrak{C}$.

Lemma 4.1.2 Let $N \geq 1$ and L be an MSC language. Then, L is the language of a ($\forall$-bounded/finite/$\forall$-bounded and finite) N-MPA iff it is the language of an extended locally-accepting ($\forall$-bounded/finite/$\forall$-bounded and finite, respectively) N-MPA.

Proof "only if": Let $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, be an MPA. For each state $\overline{s} \in F$, introduce a global initial state running a distinct copy $\mathcal{A}(\overline{s})$ of $\mathcal{A}$ with local state spaces $S_p^{\overline{s}}$ (in the following, a copy of a local state $s \in S_p$ in $\mathcal{A}(\overline{s})$ is denoted by $s^{\overline{s}}$). The set of global final states is henceforth the cartesian product $\prod_{p \in P} \bigcup_{\overline{s} \in F} \{\overline{s}[p]^{\overline{s}}\}$. The resulting MPA is locally-accepting and, obviously, recognizes the same language as $\mathcal{A}$ without having affected the number of messages, boundedness, or finiteness properties.

"if": Let $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, S^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, be an extended MPA where F is the cartesian product $\prod_{p \in P} F_p$ of sets $F_p \subseteq S_p$. Similarly to the "only if"-case, the basic idea is to create a copy $S_p^{\overline{s}_0} = S_p \times \{\overline{s}_0\}$ of S_p for any global initial state $\overline{s}_0 \in S^{in}$. Starting in some new global initial state $\overline{s}^{in}$ and switching to some state $(s, \overline{s}_0)$ now settles for simulating a run of $\mathcal{A}$ from $\overline{s}_0$ by henceforth allowing to enter no other copy than $S_p^{\overline{s}_0}$. In a global final state, it is then checked whether the other processes agree in their choice of $\overline{s}_0$. More formally, we may have local transitions of the form $((s, \overline{s}_0), \sigma, m, (s', \overline{s}_0))$ with $\overline{s}_0 \in S^{in}$ if (s, σ, m, s') is a local transition of $\mathcal{A}$. Moreover, we add kind of initial transitions $(\overline{s}^{in}[p], \sigma, m, (s, \overline{s}_0))$ if $(\overline{s}_0[p], \sigma, m, s)$ is some p-local transition of $\mathcal{A}$ with $\overline{s}_0 \in S^{in}$. It remains to reformulate the acceptance condition. Henceforth, a state $\overline{s}$ is a global final state if there is $\overline{s}_0 \in S^{in}$ such that, for any $p \in P$, either $\overline{s}[p] = \overline{s}^{in}[p]$ and $\overline{s}_0[p] \in F_p$ or $\overline{s}[p] \in F_p \times \{\overline{s}_0\}$. Again, neither the number of messages used nor boundedness or finiteness properties have changed. Note that, as F is already finite, we also create a finite number of global final states only. $\square$

Besides determinism, the absence of deadlocks is a crucial aim when designing a distributed protocol. The study of realizability without deadlocks was conceived in [AEY00] and then continued in [Loh03]. While, to some extent, finite automata over words and asynchronous automata over traces can be assumed to be free from *deadlock* states, which cannot contribute to an accepting run anymore, finite MPAs are more complicated in this regard: in general, deadlocks cannot be avoided. Even simple finite MSC languages are inherently *non-safe*. Moreover, it

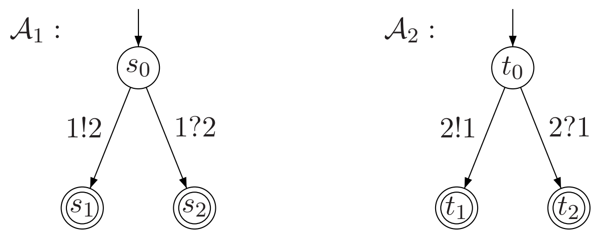


Figure 4.2: A non-safe MPA

is undecidable if an MPA has a deadlock at all. Let us be more precise and consider an MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$. We say that $(\overline{s}, \chi) \in \mathit{Conf}_{\mathcal{A}}$ is a *deadlock configuration* of $\mathcal{A}$ if there is no $(\overline{s}', \chi') \in F \times \{\chi_\varepsilon\}$ such that $(\overline{s}, \chi) \Longrightarrow^*_{\mathcal{A}} (\overline{s}', \chi')$. We call $\mathcal{A}$ *safe* if there is no deadlock configuration reachable in $\mathcal{A}$. For a set $\mathfrak{C}$ of MPAs, we denote by safe-$\mathfrak{C}$ the class of safe MPAs $\mathcal{A}$ with $\mathcal{A} \in \mathfrak{C}$. Consider the non-safe finite MPA $\mathcal{A} \in$ 1-$\forall$MPA$^f_\ell$ from Figure 4.2 (say with extra message $\circ$). It is non-safe, as the deadlock configuration $((s_1, t_1), ((1,2) \mapsto \circ, (2,1) \mapsto \circ))$ is reachable in $\mathcal{A}$. It even holds $L(\mathcal{A}) \notin \mathcal{L}$(safe-MPA), i.e., the language of $\mathcal{A}$ cannot be recognized by some safe MPA. The structural problem is that, in any implementation of $L(\mathcal{A})$ (provided there is only one global initial state), both processes 1 and 2 can independently decide to send a message, which inevitably leads into a deadlock configuration. Those phenomena are known as *non-local choice*, whose detection is studied in [BAL97] for high-level MSCs. In fact, it was shown in [GMSZ02] that any *local-choice* HMSC, which avoids non-local choice, can be implemented by some safe finite MPA. The example shows that $\mathcal{L}$(1-$\forall$MPA$^f_\ell$) and $\mathcal{L}$(safe-MPA) are incomparable wrt. inclusion. In particular, the weakest model of an MPA is able to produce inherently non-safe MSC languages.

Remark 4.1.3 The following problems are undecidable:

(a) Input: $\mathcal{A} \in$ MPA^f. Question: $L(\mathcal{A}) = \emptyset$?

(b) Input: $\mathcal{A} \in$ MPA^f. Question: Is $\mathcal{A}$ safe?

(c) Input: $\mathcal{A} \in$ MPA^f. Question: $\mathcal{A} \in \forall$MPA^f?

Proof Several decidability questions were studied for *communicating finite-state machines*, a slightly different variant of MPAs. Among them, (a problem related to) the emptiness and safety problem for communicating finite-state machines turned out to be undecidable [BZ83]. The proof can be easily adapted towards MPAs. The remaining problem is then reduced to the emptiness problem (see [Gen04] for details). $\square$

4.2 MPAs vs. Product MSC Languages

Let us identify the automata model that corresponds precisely to the class of (weak) product MSC languages. It will provide the basis for further expressiveness results in the scope of product MSC languages.

Lemma 4.2.1 ([AEY00])

$$\mathcal{P}^0 \;=\; \mathcal{L}(\text{1-MPA}_\ell)$$

Corollary 4.2.2

$$\mathcal{P} \;=\; \mathcal{L}(\text{1-MPA})$$

Proof "$\supseteq$": According to Lemma 4.1.2, a 1-MPA can be transformed into an equivalent extended locally-accepting 1-MPA $((\mathcal{A}_p)_{p\in P}, \mathcal{D}, S^{in}, F)$, which then recognizes $\bigcup_{\overline{s}\in S^{in}} L(((\mathcal{A}_p)_{p\in P}, \mathcal{D}, \overline{s}, F))$, thus, the finite union of languages that are each accepted by some locally-accepting 1-MPA. The assertion follows from Lemma 4.2.1 and Definition 3.8.1.

"$\subseteq$": Similarly, any MSC language $L \in \mathcal{P}$ is the union of finitely many languages $L_1, \ldots, L_k \in \mathcal{P}^0$, which, according to Lemma 4.2.1 are recognized by locally-accepting 1-MPAs $\mathcal{A}^1, \ldots, \mathcal{A}^k$ (each employing, say, $\circ$ as synchronization message) with global initial states $\overline{s}_1, \ldots, \overline{s}_k$ and sets of global final states $F^1, \ldots, F^k$, respectively, where, for each $i \in \{1, \ldots, k\}$, $F^i = \prod_{p\in P} F^i_p$ for some $F^i_p \subseteq S^i_p$ (let hereby S^i_p be the set of p-local states of $\mathcal{A}^i$). Without loss of generality, $\mathcal{A}^1, \ldots, \mathcal{A}^k$ have mutually distinct local state spaces. The extended locally-accepting 1-MPA recognizing L processwise merges the state spaces and transitions of $\mathcal{A}^1, \ldots, \mathcal{A}^k$, employs $\{\overline{s}_1, \ldots, \overline{s}_k\}$ being the set of global initial states, and, similarly to the proof of Lemma 4.1.2, sets the set of global final states to be $\prod_{p\in P} \bigcup_{i\in\{1,\ldots,k\}} F^i_p$. The assertion then follows from Lemma 4.1.2. $\square$

4.3 MPAs vs. Regular MSC Languages

Henriksen et al. provide an automata-theoretic characterization of the class of regular MSC languages in terms of (strongly-)$\forall$-bounded finite MPAs.

Theorem 4.3.1 ([HMKT00b])

$$\mathcal{R} \;=\; \mathcal{L}(\forall\text{MPA}^f) \;=\; \mathcal{L}(\forall_!\text{MPA}^f)$$

In the framework of regular MSC languages, restricting to one synchronization message (and a local acceptance condition) on the automata side then means to restrict to (weak, respectively) product languages:

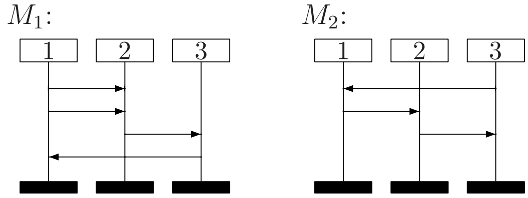


Figure 4.3: Universal boundedness vs. strong boundedness

Lemma 4.3.2

$$\mathcal{RP}^0 \;=\; \mathcal{L}(1\text{-}\forall\mathrm{MPA}^f_\ell)$$

Proof "$\supseteq$": According to Lemma 4.2.1, the language of an MPA $\mathcal{A} \in 1\text{-}\forall\mathrm{MPA}^f_\ell$ is a weak product language, and, according to Theorem 4.3.1, $L_w(\mathcal{A}) = Lin(L(\mathcal{A}))$ constitutes a regular word language over Act.

"$\subseteq$": Let $L \in \mathcal{RP}^0$ and, for $p \in P$, $\mathcal{A}_p = (S_p, \Delta_p, \overline{s}^{in}_p, F_p)$ be a finite automaton over Act_p satisfying $L(\mathcal{A}_p) = L \upharpoonright p := \{M \upharpoonright p \mid M \in L\}$. Consider the MPA $\mathcal{A} = ((\mathcal{A}'_p)_{p\in P}, \mathcal{D}, \overline{s}^{in}, F)$ with $\mathcal{D} = \{\circ\}$, $\overline{s}^{in} = (\overline{s}^{in}_p)_{p\in P}$, $F = \prod_{p\in P} F_p$, and $\mathcal{A}'_p = (S_p, \Delta'_p)$ where, for any $s, s' \in S_p$ and $\sigma \in Act_p$, $(s, \sigma, \circ, s') \in \Delta'_p$ if $(s, \sigma, s') \in \Delta_p$. We claim that both $\mathcal{A} \in 1\text{-}\forall\mathrm{MPA}^f_\ell$ and $L(\mathcal{A}) = L$. First, it is easy to see that $L \subseteq L(\mathcal{A})$. Now assume an MSC M to be contained in $L(\mathcal{A})$. For each $p \in P$, $M \upharpoonright p \in L(\mathcal{A}_p) = L \upharpoonright p$ so that there is an MSC $M' \in L$ with $M' \upharpoonright p = M \upharpoonright p$. From the definition of $\mathcal{P}^0$, it then immediately follows that M is contained in L, too. Clearly, $\mathcal{A}$ is finite, locally-accepting, and $\forall$-bounded. $\square$

Note that Lemma 4.3.2 does not hold if we adopt the definition of strong universal boundedness proposed by Henriksen et al. (while Theorem 4.3.1 still holds).

Lemma 4.3.3

$$\mathcal{RP}^0 \;\supsetneqq\; \mathcal{L}(1\text{-}\forall_!\mathrm{MPA}^f_\ell)$$

Proof It remains to show strictness. Let $L = \{M_1\}^* \cup \{M_2\}^*$ with M_1 and M_2 given by Figure 4.3, and suppose there is an MPA $\mathcal{A} \in 1\text{-}\mathrm{MPA}^f_\ell$ with $L(\mathcal{A}) = L$. Then, for each natural $n \geq 1$, the word

$$(1!2)^2\left((3!1)(1?3)(1!2)^2(2?1)(2!3)(3?2)\right)^n \in \mathbb{W}(Act)$$

leads from the initial configuration of $\mathcal{A}$ via $\Longrightarrow_\mathcal{A}$ to some configuration (s, χ) with $\chi((1,2)) = n + 3$. Thus, $\mathcal{A}$ cannot be strongly-$\forall$-bounded. Nevertheless, L is contained in $\mathcal{RP}^0$ and $\forall 2$-bounded. $\square$

However, if we restrict to safely realizable MSC languages, Lemma 4.3.2 holds for both definitions of universal boundedness, as safely realizable languages can be realized by MPAs in which no deadlock configuration is reachable, i.e., we need not pay attention to configurations that do not contribute to the recognized language anyway.

Corollary 4.3.4 $\mathcal{RP} = \mathcal{L}(1\text{-}\forall\text{MPA}^f)$

The proof of Corollary 4.3.4 is analogous to the one of Corollary 4.2.2.

Let us in the following compare traces and MSCs in the scope of regular MSC languages and justify that, in this regard, we have chosen the same terminology for traces and MSCs. In particular, we raise the hope that results and logics regarding product trace languages are amenable to MSCs, such as the local temporal logic PTL, which is tailored to systems that support product behavior [Thi95]. Though the following investigation is rather independent of MPAs, it allows to immediately derive corresponding expressiveness results for automata.

Proposition 4.3.5 ([Kus02, HMKT99]) For any $B \geq 1$ and any $L \subseteq \mathbb{MSC}_{\forall B}$,

$$L \in \mathcal{R}_{\mathbb{MSC}} \text{ iff } Tr_B(L) \in \mathcal{R}_{\mathbb{TR}(\tilde{\Sigma}_B)}.$$

Proof "only if": Let $B \geq 1$ and let $L \subseteq \mathbb{MSC}_{\forall B}$ be a B-bounded regular MSC language, i.e., $Lin(L)$ is recognized by some minimal finite automaton $\mathcal{A}$ over Act (which means that, in particular, from each state of $\mathcal{A}$, some final state is reachable). As each state of $\mathcal{A}$ can be associated with a fixed channel contents, it is easy to provide the transitions of $\mathcal{A}$ with additional labelings from $\{1, \ldots, B\}$, which leads to a finite automaton over $Act \times \{1, \ldots, B\}$ recognizing $Lin(Tr_B(L))$.

"if": As shown in [HMKT99], any asynchronous automaton over $\tilde{\Sigma}_B$ has an equivalent counterpart in form of a (strongly-)bounded finite MPA. Together with Theorems 2.4.10 and 4.3.1, this proves the lemma. ☐

Proposition 4.3.6 For any $B \geq 1$ and any $L \subseteq \mathbb{MSC}_{\forall B}$,

$$L \in \mathcal{RP}^0_{\mathbb{MSC}} \text{ iff } Tr_B(L) \in \mathcal{RP}^0_{\mathbb{TR}(\tilde{\Sigma}_B)}.$$

Proof According to Proposition 4.3.5, the operator Tr_B and its converse both preserve regularity.

"only if": Suppose $L \subseteq \mathbb{MSC}_{\forall B}$ to be a weak regular product MSC language. Recall that $Tr_B(L)$ is a regular trace language over $\tilde{\Sigma}_B = (\overline{Act_\gamma})_{\gamma \in P \cup Co}$. Moreover, let $T \in \mathbb{TR}(\tilde{\Sigma}_B)$ such that, for any $\gamma \in P \cup Co$, there is a trace $T_\gamma \in Tr_B(L)$ satisfying $T_\gamma \restriction \gamma = T \restriction \gamma$. Then, $T \in Tr_B(\mathbb{MSC}_{\forall B})$ and, in particular, we have $T_p \restriction p = T \restriction p$ and, thus, $Tr_B^{-1}(T_p) \restriction p = Tr_B^{-1}(T) \restriction p$ for any $p \in P$, which implies $Tr_B^{-1}(T) \in L$ and $T \in Tr_B(L)$.

"if": Suppose $L \subseteq \mathbb{MSC}_{\forall B}$ to generate a weak regular trace language over $\tilde{\Sigma}_B$, i.e., $Tr_B(L) \in \mathcal{RP}^0_{\mathbb{TR}(\tilde{\Sigma}_B)}$, and let $M \in \mathbb{MSC}_{\forall B}$ such that, for any $p \in P$, there is $M_p \in L$ with $M_p \upharpoonright p = M \upharpoonright p$. Trivially, we have that, for any $p \in P$, $Tr_B(M_p) \upharpoonright p = Tr_B(M) \upharpoonright p$. Moreover, for any $\gamma = (p!q, q?p, n) \in Co$, $Tr_B(M_p) \upharpoonright \gamma = Tr_B(M) \upharpoonright \gamma$ (note that also $Tr_B(M_q) \upharpoonright \gamma = Tr_B(M) \upharpoonright \gamma$). This is because, in the trace of a $\forall B$-bounded MSC, the n-th receipt of a message through (p, q) is ordered before sending a message from p to q for the $(n + B)$-th time. Altogether, we have $Tr_B(M) \in Tr_B(L)$ and, consequently, $M \in L$. □

The following extension towards finite unions of weak regular product languages is obvious.

Corollary 4.3.7 For any $B \geq 1$ and any $L \subseteq \mathbb{MSC}_{\forall B}$,

$$L \in \mathcal{RP}_{\mathbb{MSC}} \text{ iff } Tr_B(L) \in \mathcal{RP}_{\mathbb{TR}(\tilde{\Sigma}_B)}.$$

4.4 MPAs vs. EMSO-definable MSC Languages

We now turn towards one of our main results, which establishes the relationship between finite MPAs and EMSO logic over MSCs.

4.4.1 $\mathcal{MPA} = \mathcal{EMSO}_{\mathbb{MSC}}$

In fact, any EMSO-definable MSC language is implementable as a finite MPA and, vice versa, any MSC language recognized by some finite MPA has an appropriate EMSO counterpart.

Theorem 4.4.1

$$\mathcal{MPA} = \mathcal{EMSO}_{\mathbb{MSC}}$$

The theorem follows directly from Lemma 4.4.3 and Lemma 4.4.4, which will be shown in the following sections. From Lemma 4.4.3, we can furthermore deduce the following:

Theorem 4.4.2 The following two problems are undecidable:

(a) Satisfiability for EMSO sentences over $\mathbb{MSC}$

(b) Universality for Σ_2-sentences over $\mathbb{MSC}$

Proof Using Remark 4.1.3 and Lemma 4.4.3, we get Corollary 4.4.2 (a). Corollary 4.4.2 (b) follows from an easy reduction from the satisfiability problem: there is an MSC satisfying a given EMSO sentence φ iff *not* any MSC satisfies the dual of φ, which can be written as a Σ_2-sentence. □

4.4.2 From MPA^f to EMSO

The easier part is to provide an EMSO formula for a given finite MPA. We hereby mainly follow similar constructions applied, for example, to finite and asynchronous automata (cf. Theorem 2.3.3 and [DGK00] for examples).

Lemma 4.4.3 $\mathcal{MPA} \subseteq \mathcal{EMSO}_{\mathbb{MSC}}$

Proof Let $N \geq 1$ and let $\mathcal{A} = ((\mathcal{A}_p)_{p\in P}, \{1, \ldots, N\}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, be a finite MPA. We assume $F \neq \emptyset$. Note that $\mathcal{A}_p$, once equipped with an initial state $s \in S_p$ and a set of final states $F_p \subseteq S_p$, can be considered as a finite word automaton generating a regular word language over $Act_p \times \{1, \ldots, N\}$. In case that $N = 1$, we can even understand $\mathcal{A}_p$ to recognize a word language over Act_p ignoring the respective message component in the transition relation Δ_p.

Our aim is to exhibit an EMSO(Act, P_{c})-sentence Ψ such that $L(\Psi) = L(\mathcal{A})$. Recall that the class of word languages that are EMSO$_{\mathbb{W}}$-definable coincides with the class of regular word languages. Clearly, the language of $\mathcal{A}$ equals $\bigcup_{\overline{s}\in F} L(((\mathcal{A}_p)_{p\in P}, \{1, \ldots, N\}, \overline{s}^{in}, \{\overline{s}\}))$. So let, for each global final state $\overline{s} \in F$ and each process $p \in P$,

$$\varphi^{\overline{s},p} = \exists X_1^{\overline{s},p} \ldots \exists X_{n_{\overline{s},p}}^{\overline{s},p} \psi^{\overline{s},p}(X_1^{\overline{s},p}, \ldots, X_{n_{\overline{s},p}}^{\overline{s},p})$$

be an EMSO$(Act_p, -)$-sentence with first-order kernel $\psi^{\overline{s},p}$ such that $L_{\mathbb{W}(Act_p)}(\varphi^{\overline{s},p})$ is the language of the finite automaton $\mathcal{A}_p$ with initial state $\overline{s}^{in}[p]$ and set of final states $\{\overline{s}[p]\}$. The formula Ψ now requires the existence of an assignment of synchronization messages to the events that, on the one hand, respects that communicating events have to be equally labeled and, on the other hand, meets the restrictions imposed by the formulas $\varphi^{\overline{s},p}$ for at least one final state $\overline{s} \in F$. It is given by

$$\begin{array}{rcl}\Psi & = & \exists X_1 \ldots \exists X_N \; \exists \overline{X} \\ & & \Big(\quad \text{partition}(X_1, \ldots, X_N) \\ & & \;\wedge \;\; \text{consistent}(X_1, \ldots, X_N) \\ & & \;\wedge \;\; \bigvee_{\overline{s}\in F} \bigwedge_{p\in P} \text{process}_{\overline{s},p}(X_1, \ldots, X_N, X_1^{\overline{s},p}, \ldots, X_{n_{\overline{s},p}}^{\overline{s},p}) \; \Big)\end{array}$$

where $\overline{X}$ is the block of all the second-order variables $X_i^{\overline{s},p}$.

The formula partition$(X_1, \ldots, X_N)$ ensures that the variables $X_1, \ldots, X_N$ define a mapping from the set of events of an MSC to the set of synchronization messages $\{1, \ldots, N\}$. Then, consistent$(X_1, \ldots, X_N)$ guarantees that the mapping is consistent, i.e., a send event and its corresponding receive event are equally labeled wrt. the alphabet of synchronization messages:

$$\text{consistent}(X_1, \ldots, X_N) = \forall x \forall y \left(x \lhd_{\mathrm{c}} y \rightarrow \bigvee_{i \in \{1,\ldots,N\}} (x \in X_i \wedge y \in X_i) \right)$$

For $\overline{s} \in F$ and $p \in P$, the formula $\text{process}_{\overline{s},p}(X_1, \ldots, X_N, X_1^{\overline{s},p}, \ldots, X_{n_{\overline{s},p}}^{\overline{s},p})$ ensures that the total order that process p induces wrt. an MSC corresponds to what the word automaton $\mathcal{A}_p$ with initial state $\overline{s}^{in}[p]$ and set of final states $\{\overline{s}[p]\}$ recognizes. It is the projection $\|\psi^{\overline{s},p}\|_p$ of the formula $\psi^{\overline{s},p}$ on p, which is inductively derived. In particular (recall that $\psi^{\overline{s},p}$ is the *first-order* kernel of a formula interpreted over words $(E, \lhd, \lambda) \in \mathbb{W}(Act_p \times \{1, \ldots, N\})$ with $\lambda : E \rightarrow Act_p \times \{1, \ldots, N\}$),

- $\|\lambda(x) = (\sigma, n)\|_p = (\lambda(x) = \sigma) \wedge x \in X_n$,
- $\|x \in X\|_p = x \in X$,
- $\|x \lhd y\|_p = x \lhd_p y$,
- $\|\exists x \varphi\|_p = \exists x(P(x) = p \wedge \|\varphi\|_p)$, and
- $\|\forall x \varphi\|_p = \forall x(P(x) = p \rightarrow \|\varphi\|_p)$,

while the remaining derivations are defined canonically. Note that, though Ψ allows the set variables $X_i^{\overline{s},p}$ to range over arbitrary subsets of events, in $\|\psi^{\overline{s},p}\|_p$, their interpretation is restricted to elements of process p. □

4.4.3 From EMSO to MPA^f

We now show that an EMSO(Act, P_{c})-sentence that is interpreted over MSCs can be transformed into an equivalent finite MPA.

Lemma 4.4.4

$$\mathcal{MPA} \supseteq \mathcal{EMSO}_{\mathbb{MSC}}$$

Proof Suppose φ to be an EMSO(Act, P_{c})-sentence. As $\mathbb{MSC}$ is a set of bounded degree, we can, according to Theorem 2.2.13, assume the existence of a graph acceptor $\mathcal{B}$ over (Act, P_{c}) that, running on MSCs, recognizes the MSC language defined by φ. In turn, $\mathcal{B}$ will be translated into a finite MPA $\mathcal{A}$ that captures the application of $\mathcal{B}$ to MSCs, i.e., $L(\mathcal{A}) = L_{\mathbb{MSC}}(\mathcal{B})$. So let $\mathcal{B} = (Q, R, \mathfrak{S}, Occ)$ be a graph acceptor over (Act, P_{c}). (A simple graph acceptor tailored to MSCs—without occurrence constraints and a singleton as set of states—and a corresponding run are depicted in Figure 4.5 on the facing page and Figure 4.6 on page 76, respectively. The recognized MSC language is $\{G\}^*$ where G is taken from Figure 3.9 on page 52. However, there is an equivalent graph acceptor even with radius 0. To verify this is left as an exercise to the reader.)

For our purpose, it suffices to consider only those R-spheres $H \in \mathfrak{S}$ for which there is an extended MSC $M = (E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda) \in \mathbb{MSC}^Q$, which has an extended labeling function $\lambda : E \to Act \times Q$, and an event $e \in E$ such that H is the R-sphere of M around e. Other spheres cannot contribute to an MSC. Because, to become part of a run on some MSC M, an R-sphere has to admit an embedding into M. Accordingly, the 2-sphere illustrated in part (a) of Figure 4.4 on the next page may contribute to a run on an MSC (it can be complemented by a 1!3-labeled event arranged in order between the two other events of process 1), while the 2-sphere illustrated aside is irrelevant for accepting MSCs and will be ignored in the following. This assumption is essential, as it ensures that, for each $H = (E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda, \gamma) \in \mathfrak{S}$ and $e \in E$, $d_H(e, \gamma) < R$ implies that E also contains a communication partner of e wrt. $\lhd_{\mathrm{c}}$.

In the following, we use notions that we have introduced for MSCs also for spheres $(E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda, \gamma)$ over $(Act \times Q, P_{\mathrm{c}})$, such as $P(e)$ and E_p (to indicate the process of $e \in E$ and as abbreviation for $\lambda^{-1}(Act_p \times Q)$, respectively). Note also that, wrt. spheres, $\leq_p$ is not necessarily a total order. For example, considering the 2-sphere from Figure 4.4 (a), $P(\mathrm{a}) = 1$, $E_1 = \{\mathrm{a}, \mathrm{e}\}$, and $\mathrm{b} \leq_2 \mathrm{d}$, but *not* $\mathrm{a} \leq_1 \mathrm{e}$. Let $maxE := \max\{|E| \mid (E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda, \gamma) \in \mathfrak{S}\}$ and let $\mathfrak{S}^+$ be the set of *extended R-spheres*, i.e., the set of structures $((E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda, \gamma, e), i)$ where $(E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda, \gamma) \in \mathfrak{S}$, $e \in E$ is the *active node*, and $i \in \{1, \ldots, 4 \cdot maxE^2 + 1\}$ is the current *instance*. For $p \in P$, we define $\mathfrak{S}_p := \{(E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda, \gamma) \in \mathfrak{S} \mid P(\gamma) = p\}$ and, furthermore, $\mathfrak{S}_p^+ := \{((E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda, \gamma, e), i) \in \mathfrak{S}^+ \mid P(e) = p\}$. Finally, let $\max(Occ)$ denote the least threshold n such that Occ does not distinguish occurrence numbers $\geq n$. For readability, we let in the following $\lhd$ denote the collection $(\{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}})$ and just write $(E, \lhd, \lambda, \gamma)$ instead of $(E, \{\lhd_p\}_{p \in P}, \lhd_{\mathrm{c}}, \lambda, \gamma)$.

The idea of the transformation is that, roughly speaking, $\mathcal{A}$ guesses a tiling of the MSC to be read and then verifies that the tiling corresponds to an accepting run of $\mathcal{B}$. Accordingly, a local state of $\mathcal{A}$ holds a set of *active* R-spheres, i.e., a set of spheres that play a role in its immediate environment of distance at most R. Each local state s (apart from the initial states, as we will see) carries exactly one extended R-sphere $((E, \lhd, \lambda, \gamma, e), i) \in \mathfrak{S}^+$ with $\gamma = e$, which means that a run of $\mathcal{B}$ assigns the state associated with γ to the event that cor-

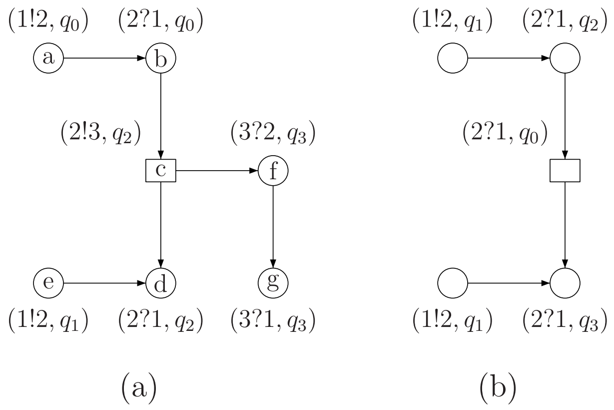


Figure 4.4: The sphere(s) of a graph acceptor over (Act, P_{c})

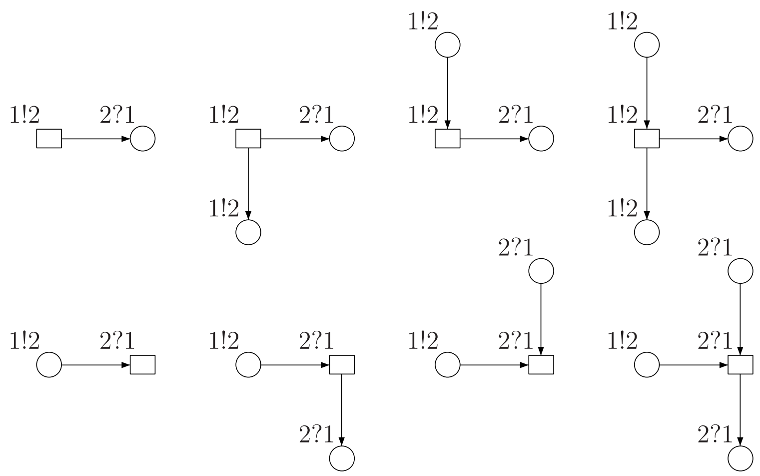


Figure 4.5: A graph acceptor over (Act, P_{c})

responds to s. To establish isomorphism between $(E, \lhd, \lambda, \gamma)$ and the R-sphere induced by s, s transfers/obtains its obligations in form of an extended R-sphere $((E, \lhd, \lambda, \gamma, e'), i)$ to/from its immediate neighbors, respectively. For example, provided e is labeled with a send action and there is $e' \in E$ with $e \lhd_{\mathrm{c}} e'$, the message to be sent when entering state s will contain $((E, \lhd, \lambda, \gamma, e'), i)$, which, in turn, the receiving process understands as a requirement to be satisfied. As there may be an overlapping of isomorphic R-spheres, a state can hold several instances of one and the same sphere, which then refer to distinct states/events as corresponding sphere centers. Those instances will be distinguished by means of the natural i. The benefit of i will become clearer before long.

Let us turn to the construction of $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, which is given as follows: For $p \in P$, a local state of $\mathcal{A}_p$ is a pair $(\mathcal{S}, \nu)$ where

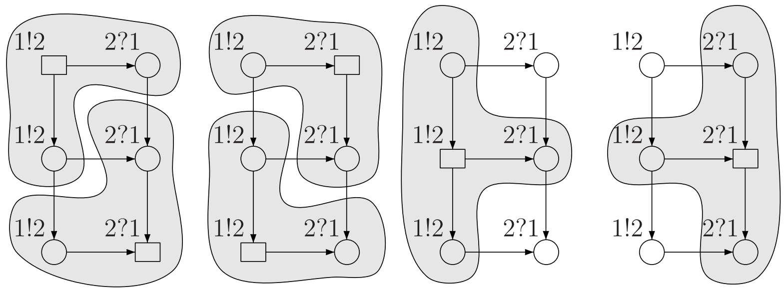


Figure 4.6: The run of a graph acceptor

– ν is a mapping $\mathfrak{S}_p \to \{0, \ldots, \max(Occ)\}$
(let in the following ν_p^0 denote the function that maps each R-sphere $H \in \mathfrak{S}_p$ to 0) and

– $\mathcal{S}$ is either the empty set or it is a subset of $\mathfrak{S}_p^+$ such that

- there is exactly one extended R-sphere $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}$ with $\gamma = e$ (whose component $(E, \lhd, \lambda, \gamma)$ we identify by $\varsigma(\mathcal{S})$ from now on) and
- for any two $((E, \lhd, \lambda, \gamma, e), i), ((E', \lhd', \lambda', \gamma', e'), i') \in \mathcal{S}$,
  (a) $\lambda(e) = \lambda'(e') \in Act_p \times Q$ (so that we can assign a well-defined unique label $\lambda(\mathcal{S}) \in Act_p \times Q$ to $\mathcal{S}$, namely the labeling $\lambda(e)$ for some $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}$) and
  (b) if $(E, \lhd, \lambda, \gamma) \cong (E', \lhd', \lambda', \gamma')$ and $i = i'$, then $e = e'$.

The set $\mathcal{D}$ of synchronization messages is the cartesian product $2^{\mathfrak{S}^+} \times 2^{\mathfrak{S}^+}$. Roughly speaking, the first component of a message contains obligations the receiving state/event has to satisfy, while the second component imposes requirements that must *not* be satisfied by the receiving process to ensure isomorphism. We now turn towards the definition of Δ_p and define $((\mathcal{S}, \nu), \sigma, (\mathcal{P}, \mathcal{N}), (\mathcal{S}', \nu')) \in \Delta_p$ if the following hold:

1. $\lambda(\mathcal{S}') = (\sigma, q)$ for some $q \in Q$.

2. For any $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}$ and $e' \in E_p$, if $((E, \lhd, \lambda, \gamma, e'), i) \in \mathcal{S}'$, then $e \lhd_p e'$.

3. For any $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}'$, if $\mathcal{S} \neq \emptyset$ and e is minimal in $(E_p, \leq_p)$, then $d(e, \gamma) = R$.

4. For any $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}$, if e is maximal in $(E_p, \leq_p)$, then $d(e, \gamma) = R$.

5. For any $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}'$, if e is not minimal in $(E_p, \leq_p)$, then we have $((E, \lhd, \lambda, \gamma, e^-), i) \in \mathcal{S}$ where $e^- \in E_p$ is the unique event satisfying $e^- \lhd_p e$.

6. For any $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}$, if e is not maximal in $(E_p, \leq_p)$, then we have $((E, \lhd, \lambda, \gamma, e^+), i) \in \mathcal{S}'$ where $e^+ \in E_p$ is the unique event such that $e \lhd_p e^+$.

7. (i) In case that $\sigma = p!q$ for some $q \in P$:
   (a) for any $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}'$ and any $e' \in E$, if $e \lhd_c e'$, then we have $((E, \lhd, \lambda, \gamma, e'), i) \in \mathcal{P}$,
   (b) for any $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}'$ and any $e' \in E$, if $e \not\lhd_c e'$, then we have $((E, \lhd, \lambda, \gamma, e'), i) \in \mathcal{N}$, and
   (c) for any $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{P}$, there is $e' \in E$ such that $e' \lhd_c e$ and $((E, \lhd, \lambda, \gamma, e'), i) \in \mathcal{S}'$.

   (ii) In case that $\sigma = p?q$ for some $q \in P$:
   (a) $\mathcal{P} \subseteq \mathcal{S}'$,
   (b) $\mathcal{N} \cap \mathcal{S}' = \emptyset$, and
   (c) for any $((E, \lhd, \lambda, \gamma, e'), i) \in \mathcal{S}'$, if there is $e \in E$ with $e \lhd_c e'$, then $((E, \lhd, \lambda, \gamma, e'), i) \in \mathcal{P}$.

8. $\nu' = \nu[\varsigma(\mathcal{S}')/\min\{\nu(\varsigma(\mathcal{S}')) + 1, \max(Occ)\}]$ (i.e., ν' maps $\varsigma(\mathcal{S}')$ to the minimum of $\nu(\varsigma(\mathcal{S}')) + 1$ and $\max(Occ)$ and, otherwise, coincides with ν).

Thus, Condition 1. guarantees that any state within a run has the same labeling as the event it is assigned to. Condition 2. makes sure that, whenever there is a $\lhd_p$-edge in the input MSC, then there is a corresponding edge in the extended sphere that is passed from the source to the target state of the corresponding transition. Conversely, if there is no $\lhd_p$-edge between two nodes in the extended sphere, then it must not be passed directly to impose the same behavior on the MSC, i.e., the corresponding events in the MSC must not touch each other. Conditions 3. and, dually, 4. make sure that a sphere that does not make use of the whole radius R is employed in the initial or final phase of a run, only. By Conditions 5. and 6., extended spheres must be passed along a process line as far as possible, hereby starting in a minimal and ending in a maximal active node. Condition 7. ensures the corresponding beyond process lines, i.e., for messages. Finally, Condition 8. guarantees that the second component of each state correctly keeps track of the number of spheres used so far.

Furthermore, $\overline{s}^{in} = ((\emptyset, \nu_p^0))_{p \in P}$ and, for $(\mathcal{S}_p, \nu_p) \in S_p$, $((\mathcal{S}_p, \nu_p))_{p \in P} \in F$ if the union of mappings ν_p satisfies the requirements imposed by Occ and, for all $p \in P$ and $((E, \lhd, \lambda, \gamma, e), i) \in \mathcal{S}_p$, e is maximal in $(E_p, \leq_p)$.

In fact, it holds $L(\mathcal{A}) = L_{\mathbb{MSC}}(\mathcal{B})$:

Let $\rho : \widetilde{E} \to Q$ be an accepting run of $\mathcal{B}$ on $M = (\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_\mathrm{c}, \widetilde{\lambda}) \in \mathbb{MSC}$ and let $\widehat{\rho}$ denote the mapping $\widetilde{E} \to \mathfrak{S}$ that maps an event $e \in \widetilde{E}$ onto the R-sphere of $(\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_\mathrm{c}, (\widetilde{\lambda}, \rho))$ around e. In an accepting run (r, m) of $\mathcal{A}$ on M, r basically assigns to an event $e \in \widetilde{E}$—apart from the obvious mapping ν—the set of those extended spheres $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathfrak{S}^+$ such that there is an event $e' \in \widetilde{E}$ with both $d_M(e', e) \leq R$ and $(E, \lhd, \lambda, \gamma, e_0)$ is isomorphic to $(\widehat{\rho}(e'), e)$. Hereby, $4 \cdot maxE^2 + 1$ is sufficiently large to guarantee an instance labeling that is consistent with the transition relation of $\mathcal{A}$. If we suppose $m : \widetilde{\lhd}_\mathrm{c} \to \mathcal{D}$ to map a pair $(e_s, e_r) \in \widetilde{\lhd}_\mathrm{c}$ onto $(\mathcal{P}, \mathcal{N})$ where (set $(\mathcal{S}, \nu)$ to be $r(e_s)$) $\mathcal{P} = \{((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathfrak{S}^+ \mid$ there is $e_0 \in E$ with $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ and $e_0 \lhd_\mathrm{c} e_0'\}$ and $\mathcal{N} = \{((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathfrak{S}^+ \mid$ there is $e_0 \in E$ such that $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ and $e_0 \not\lhd_\mathrm{c} e_0'\}$, (r, m) is an accepting run of $\mathcal{A}$ on M.

Conversely, let (r, m) be an accepting run of $\mathcal{A}$ on $M = (E, \{\lhd_p\}_{p \in P}, \lhd_\mathrm{c}, \lambda) \in \mathbb{MSC}$. If we define $\rho : E \to Q$ to map an event $e \in E$ to the control state that is associated with the sphere center of $\varsigma(\mathcal{S})$ where $r(e) = (\mathcal{S}, \nu)$ for some ν, then ρ turns out to be an accepting run of $\mathcal{B}$ on M.

Let us be more precise:

Claim 4.4.5 $L_{\mathbb{MSC}}(\mathcal{B}) \subseteq L(\mathcal{A})$

Proof of Claim 4.4.5. Let $\rho : \widetilde{E} \to Q$ be an accepting run of $\mathcal{B}$ on the MSC $M = (\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_\mathrm{c}, \widetilde{\lambda}) \in \mathbb{MSC}$ and let in the following $\widetilde{\lhd}$ denote $\widetilde{\lhd}_\mathrm{c} \cup \bigcup_{p \in P} \widetilde{\lhd}_p$ and $\widehat{\rho}$ stand for the mapping $\widetilde{E} \to \mathfrak{S}$ that maps an event $e \in \widetilde{E}$ onto the R-sphere of $(\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_\mathrm{c}, (\widetilde{\lambda}, \rho))$ around e. We show that there is an accepting run of $\mathcal{A}$ on M.

Consider Figure 4.7 on the facing page, which depicts an MSC inducing two isomorphic spheres, say of type H. Obviously, e_0' is actually not allowed to carry H forward. As the example shows, however, both e_0 and e_0' must be able to carry distinct copies of H as long as they defer to distinct events of the MSC at hand as sphere centers. This is accomplished by enabling a state to carry even controversial spheres, which are then equipped with distinct instances deferring to distinct events as sphere centers. The following claim states that an assignment of instances, which resolves such a conflict and where the number of required instances only depends on $\mathcal{B}$, is always possible.

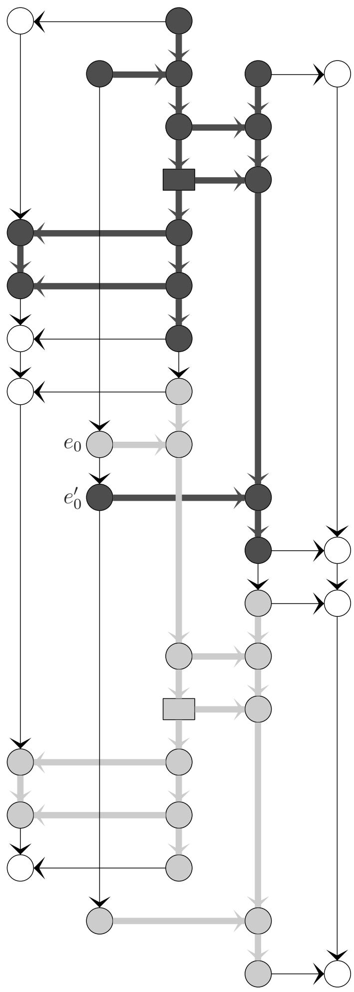


Figure 4.7: Why we need different instances of extended spheres

Claim 4.4.6 There is a mapping $\chi_{M,\rho} : \widetilde{E} \to \{1, \ldots, 4 \cdot maxE^2 + 1\}$ such that, for any $e, e', e_0, e_0' \in \widetilde{E}$ with $\widehat{\rho}(e) \cong \widehat{\rho}(e')$, $e \neq e'$, $d(e_0, e) \leq R$, and $d(e_0', e') \leq R$, if $e_0 \mathrel{\widetilde{\lhd}} e_0'$ or $e_0' \mathrel{\widetilde{\lhd}} e_0$ or $e_0 = e_0'$, then $\chi_{M,\rho}(e) \neq \chi_{M,\rho}(e')$.

Proof of Claim 4.4.6. We can reduce the existence of $\chi_{M,\rho}$ to the existence of a graph coloring. Recall some basic definitions: A *graph* G is a structure $(V, Arcs)$ where V is a finite set of *vertices* and $Arcs \subseteq V \times V$ is a set of *arcs*. For a natural $n \geq 1$, a graph $G = (V, Arcs)$ is called *n-colorable* if there is a mapping $\chi : V \to \{1, \ldots, n\}$ such that $(u, v) \in Arcs$ implies $\chi(u) \neq \chi(v)$ for any two nodes $u, v \in V$ (we then say that G is *n-colored* by χ). Furthermore, for $d \in \mathbb{N}$, G is said to be of *degree* d if $d = \max\{|Arcs(u)| \mid u \in V\}$ where, for $u \in V$, $Arcs(u) = \{v \in V \mid (u, v) \in Arcs \text{ or } (v, u) \in Arcs\}$. It is easy to show that, for any $d \in \mathbb{N}$ and any graph G of degree d without self-loops, G is $(d+1)$-colorable. The mapping $\chi_{M,\rho}$ can now be obtained as follows: Let G be the graph $(\widetilde{E}, Arcs)$ where, for any $e, e' \in \widetilde{E}$, $(e, e') \in Arcs$ iff $e \neq e'$, $\widehat{\rho}(e) \cong \widehat{\rho}(e')$, and there is $e_0, e_0' \in \widetilde{E}$ with $d(e_0, e) \leq R$, $d(e_0', e') \leq R$, and ($e_0 \mathrel{\widetilde{\lhd}} e_0'$ or $e_0' \mathrel{\widetilde{\lhd}} e_0$ or $e_0 = e_0'$). As G cannot be of degree greater than $4 \cdot maxE^2$ (for each $e \in \widetilde{E}$, there are at most four distinct events $e' \in \widetilde{E}$ such that $e \mathrel{\widetilde{\lhd}} e'$, $e' \mathrel{\widetilde{\lhd}} e$, or $e = e'$), it can be $4 \cdot maxE^2 + 1$-colored by some mapping $\chi : \widetilde{E} \to \{1, \ldots, 4 \cdot maxE^2 + 1\}$. Now set $\chi_{M,\rho}$ to be χ. This concludes the proof of Claim 4.4.6. ∎

Now let $\chi_{M,\rho}$ be the mapping from the above construction. For $H \in \mathfrak{S}$ and $e \in \widetilde{E}$, let furthermore $le_M(H, e) = |\{e' \in \widetilde{E}_{P(e)} \mid e' \mathrel{\widetilde{\leq}_{P(e)}} e,\ H \cong \widehat{\rho}(e')\}|$ and the mapping $r : \widetilde{E} \to \bigcup_{p \in P} S_p$ be given as follows: for $e \in \widetilde{E}$, we define $r(e) = (\mathcal{S}, \nu)$ where

1. $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ iff there is an event $e' \in \widetilde{E}$ such that $d(e', e) \leq R$, $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e'), e)$, and $i = \chi_{M,\rho}(e')$, and
2. for $H \in \mathfrak{S}_{P(e)}$, $\nu(H) = \min\{le_M(H, e), \max(Occ)\}$.

For $e \in \widetilde{E}$, we first verify that, in fact, $r(e) = (\mathcal{S}, \nu)$ is a valid state of $\mathcal{A}$. So suppose there are extended R-spheres $((E, \lhd, \lambda, \gamma, e_0), i), ((E', \lhd', \lambda', \gamma', e_0'), i') \in \mathcal{S}$. Of course, it holds $\lambda(e_0) = \lambda'(e_0')$. Assume now that both $\gamma = e_0$ and $\gamma' = e_0'$. But then the requirements $(E, \lhd, \lambda, \gamma, \gamma) \cong (\widehat{\rho}(e), e)$ and $(E', \lhd', \lambda', \gamma', \gamma') \cong (\widehat{\rho}(e), e)$ imply $(E, \lhd, \lambda, \gamma, \gamma) \cong (E', \lhd', \lambda', \gamma', \gamma')$. In particular, it holds $\varsigma(\mathcal{S}) = (E, \lhd, \lambda, \gamma) \cong \widehat{\rho}(e)$. Furthermore, $i = i' = \chi_{M,\rho}(e)$. Now assume $(E, \lhd, \lambda, \gamma) \cong (E', \lhd', \lambda', \gamma')$ and $i = i'$. There are events $e_1, e_2 \in \widetilde{E}$ such that $d(e_1, e) \leq R$, $d(e_2, e) \leq R$, $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e_1), e)$, $(E, \lhd, \lambda, \gamma, e_0') \cong (\widehat{\rho}(e_2), e)$, and $i = \chi_{M,\rho}(e_1) = \chi_{M,\rho}(e_2)$. Clearly, we have $\widehat{\rho}(e_1) \cong \widehat{\rho}(e_2)$. Furthermore, $e_1 = e_2$ and, consequently, $e_0 = e_0'$. This is because $e_1 \neq e_2$, according to Claim 4.4.6, implies $\chi_{M,\rho}(e_1) \neq \chi_{M,\rho}(e_2)$, which contradicts the premise.

Let $m : \widetilde{\lhd}_c \to \mathcal{D}$ map a pair $(e_s, e_r) \in \widetilde{\lhd}_c$ onto $(\mathcal{P}, \mathcal{N})$ where (set $(\mathcal{S}, \nu)$ to be $r(e_s)$) $\mathcal{P} = \{((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathfrak{S}^+ \mid$ there is $e_0 \in E$ with $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ and $e_0 \lhd_c e_0'\}$ and $\mathcal{N} = \{((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathfrak{S}^+ \mid$ there is $e_0 \in E$ such that $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ and $e_0 \not\lhd_c e_0'\}$. In the following, we verify that (r, m) is a run of $\mathcal{A}$ on M. For any distinct processes $p, q \in P$, $e \in \widetilde{E}_p$, and $e_r \in \widetilde{E}_q$ with $e \mathrel{\widetilde{\lhd}_c} e_r$, we check that $(r^-(e), \widetilde{\lambda}(e), m((e, e_r)), r(e)) \in \Delta_p$. So set $(\mathcal{S}, \nu)$ to be $r^-(e)$ and $(\mathcal{S}', \nu')$ to be $r(e)$.

1. Of course, $\lambda(\mathcal{S}') = (\widetilde{\lambda}(e), q)$ for some $q \in Q$.

2. Let $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ and $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{S}'$ for some $e_0' \in E_p$ and let $e^- \in \widetilde{E}_p$ such that $e^- \mathrel{\widetilde{\lhd}_p} e$ (as $\mathcal{S} \neq \emptyset$, such an e^- must exist). There is $e^{-\prime}, e' \in \widetilde{E}$ such that $d(e^{-\prime}, e^-) \leq R$, $d(e', e) \leq R$, $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e^{-\prime}), e^-)$, $(E, \lhd, \lambda, \gamma, e_0') \cong (\widehat{\rho}(e'), e)$, and $i = \chi_{M,\rho}(e^{-\prime}) = \chi_{M,\rho}(e')$. We show $e^{-\prime} = e'$, as then $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e'), e^-)$, $(E, \lhd, \lambda, \gamma, e_0') \cong (\widehat{\rho}(e'), e)$, and $e^- \mathrel{\widetilde{\lhd}_p} e$ imply $e_0 \lhd_p e_0'$. But $e^{-\prime} \neq e'$, according to Claim 4.4.6, implies $\chi_{M,\rho}(e') \neq \chi_{M,\rho}(e)$, which leads to a contradiction to the above assumption.

3. Suppose $\mathcal{S} \neq \emptyset$ and suppose there is $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}'$ with e_0 minimal in $(E_p, <_p)$. There is $e' \in \widetilde{E}$ such that $d(e', e) \leq R$ and $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e'), e)$. As $\mathcal{S} \neq \emptyset$, e is not minimal in $(\widetilde{E}_p, \widetilde{<}_p)$ and, consequently, $d(\gamma, e_0) = d(e', e) = R$ (if $d(e', e) < R$, e would have to be minimal in $(\widetilde{E}_p, \widetilde{<}_p)$).

4. Let $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ with e_0 maximal in $(E_p, <_p)$ and let $e^- \in \widetilde{E}_p$ such that $e^- \mathrel{\widetilde{\lhd}_p} e$. Furthermore, as $r^-(e) = r(e^-)$, there is $e^{-\prime} \in \widetilde{E}$ such that both $d(e^{-\prime}, e^-) \leq R$ and $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e^{-\prime}), e^-)$. As e^- is not maximal in $(\widetilde{E}_p, \widetilde{<}_p)$, $d(e_0, \gamma) = d(e^{-\prime}, e^-) = R$ (analogously to 3., if $d(e^{-\prime}, e^-) < R$, e^- would have to be maximal in $(\widetilde{E}_p, \widetilde{<}_p)$).

5. Suppose there is an extended R-sphere $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}'$ with e_0 not minimal in $(E_p, <_p)$. Let $e_0^- \in E$ such that $e_0^- \lhd_p e_0$. As $r(e) = (\mathcal{S}', \nu')$, there is $e' \in \widetilde{E}$ with $d(e', e) \leq R$ such that $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e'), e)$ and $i = \chi_{M,\rho}(e')$. As a consequence, e is not minimal in $(\widetilde{E}_p, \widetilde{<}_p)$ so that there is $e^- \in \widetilde{E}$ with $e^- \mathrel{\widetilde{\lhd}_p} e$. As, furthermore, $d(e', e^-) = d(\gamma, e_0^-) \leq R$ and $(E, \lhd, \lambda, \gamma, e_0^-) \cong (\widehat{\rho}(e'), e^-)$, it holds $((E, \lhd, \lambda, \gamma, e_0^-), i) \in \mathcal{S}$.

6. Suppose there is an extended R-sphere $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ (then e is not minimal in $(\widetilde{E}_p, \widetilde{<}_p)$, so let $e^- \in \widetilde{E}_p$ such that $e^- \mathrel{\widetilde{\lhd}_p} e$) with e_0 not maximal in $(E_p, <_p)$. Let $e_0^+ \in E$ such that $e_0 \lhd_p e_0^+$. As we have $r^-(e) = r(e^-) = (\mathcal{S}, \nu)$, there exists $e^{-\prime} \in \widetilde{E}$ with $d(e^{-\prime}, e^-) \leq R$, $(E, \lhd, \lambda, \gamma, e_0) \cong$

$(\widehat{\rho}(e^{-\prime}), e^-)$, and $i = \chi_{M,\rho}(e^{-\prime})$. Since then $d(e^{-\prime}, e) = d(\gamma, e_0^+) \leq R$ and also $(E, \lhd, \lambda, \gamma, e_0^+) \cong (\widehat{\rho}(e^{-\prime}), e)$, we have $((E, \lhd, \lambda, \gamma, e_0^+), i) \in \mathcal{S}'$.

7. Let $\mathcal{P}, \mathcal{N} \subseteq \mathfrak{S}^+$ such that $m((e, e_r)) = (\mathcal{P}, \mathcal{N})$.

   (a) Let $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}'$ and $e_0' \in E$. According to the definition of m, $e_0 \lhd_c e_0'$ implies $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{P}$.

   (b) Let $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}'$ and $e_0' \in E$. According to the definition of m, $e_0 \not\lhd_c e_0'$ implies $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{N}$.

   (c) Let $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{P}$. Then, due to the definition of $\mathcal{P}$, there is $e_0' \in E$ with $e_0' \lhd_c e_0$ and $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{S}'$.

8. As $\varsigma(\mathcal{S}') \cong \widehat{\rho}(e)$ and $|\{e' \mathrel{\widetilde{\leq}_p} e \mid \varsigma(\mathcal{S}') \cong \widehat{\rho}(e')\}| = |\{e' \mathrel{\widetilde{<}_p} e \mid \varsigma(\mathcal{S}') \cong \widehat{\rho}(e')\}| + 1$, we have $\nu'(\varsigma(\mathcal{S}')) = \min\{|\{e' \mathrel{\widetilde{<}_p} e \mid \varsigma(\mathcal{S}') \cong \widehat{\rho}(e')\}| + 1, \max(Occ)\}$. Furthermore, $\nu'(H) = \nu(H)$ if $H \neq \varsigma(\mathcal{S}')$.

Verifying $(r^-(e), \widetilde{\lambda}(e), m((e_s, e)), r(e)) \in \Delta_p$ for any $e \in \widetilde{E}_p$ and $e_s \in \widetilde{E}$ with $e_s \mathrel{\widetilde{\lhd}_c} e$ differs from the above scheme only in point 7. (set $(\mathcal{S}, \nu)$ to be $r(e_s)$ and $(\mathcal{S}', \nu')$ to be $r(e)$ and let $\mathcal{P}, \mathcal{N} \subseteq \mathfrak{S}^+$ such that $m((e_s, e)) = (\mathcal{P}, \mathcal{N})$):

7. (a) Suppose there is $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{P}$. Then there exists $e_0 \in E$ with $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ and $e_0 \lhd_c e_0'$. Due to $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$, there is $e_s' \in \widetilde{E}$ with $d(e_s', e_s) \leq R$, $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e_s'), e_s)$, and $i = \chi_{M,\rho}(e_s')$. As then $d(e_s', e) = d(\gamma, e_0') \leq R$ and $(E, \lhd, \lambda, \gamma, e_0') \cong (\widehat{\rho}(e_s'), e)$, $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{S}'$.

   (b) Suppose there is $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{N} \cap \mathcal{S}'$. Then there is $e_0 \in E$ with $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ and $e_0 \not\lhd_c e_0'$. Due to $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$, there is $e_s' \in \widetilde{E}$ satisfying $d(e_s', e_s) \leq R$, $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e_s'), e_s)$, and $i = \chi_{M,\rho}(e_s')$. Due to $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{S}'$, there is also $e' \in \widetilde{E}$ with $d(e', e) \leq R$, $(E, \lhd, \lambda, \gamma, e_0') \cong (\widehat{\rho}(e'), e)$, and $i = \chi_{M,\rho}(e')$. Suppose $e_s' \neq e'$. But then, as $\widehat{\rho}(e_s') \cong \widehat{\rho}(e')$, $\chi_{M,\rho}(e_s') \neq \chi_{M,\rho}(e')$, which leads to a contradiction. Now suppose $e_s' = e'$. But then $e_s \mathrel{\widetilde{\lhd}_c} e$ implies $e_0 \lhd_c e_0'$, also contradicting the premise.

   (c) Suppose now there exist $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{S}'$ and $e_0 \in E$ with $e_0 \lhd_c e_0'$. Then there is an event $e' \in \widetilde{E}$ with $d(e', e) \leq R$, $(E, \lhd, \lambda, \gamma, e_0') \cong (\widehat{\rho}(e'), e)$, and $i = \chi_{M,\rho}(e')$. As we have $d(e', e_s) = d(\gamma, e_0) \leq R$ and $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e'), e_s)$, it holds $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$ and, thus, $((E, \lhd, \lambda, \gamma, e_0'), i) \in \mathcal{P}$.

In the following, we verify that (r, m) is accepting. So set, given $p \in P$, $(\mathcal{S}_p, \nu_p)$ to be $(\emptyset, \nu_p^0)$ if $\widetilde{E}_p$ is empty and, otherwise, $(\mathcal{S}_p, \nu_p)$ to be $r(e_p)$ where $e_p \in \widetilde{E}_p$

is the maximal event wrt. $\widetilde{\leq}_p$. Clearly, the union of mappings ν_p carries, for each $H \in \mathfrak{S}$, the number of occurrences of H in $\widehat{\rho}$. Furthermore, for all $p \in P$ and $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}_p$, e_0 is maximal in $(E_p, <_p)$. Because suppose there is $e'_0 \in E$ with $e_0 \lhd_p e'_0$. But then, as there exists no $e^+ \in \widetilde{E}$ satisfying $e_p \mathrel{\widetilde{\lhd}_p} e^+$, there is no $e' \in \widetilde{E}$ either with $d(e', e_p) \leq R$ such that $(E, \lhd, \lambda, \gamma, e_0) \cong (\widehat{\rho}(e'), e_p)$, which contradicts the definition of r. This concludes the proof of Claim 4.4.5. □

Claim 4.4.7 $L(\mathcal{A}) \subseteq L_{\mathbb{MSC}}(\mathcal{B})$

Proof of Claim 4.4.7. Let (r, m) be an accepting run of $\mathcal{A}$ on the MSC $M = (\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_c, \widetilde{\lambda}) \in \mathbb{MSC}$ (again, let $\widetilde{\lhd}$ denote $\widetilde{\lhd}_c \cup \bigcup_{p \in P} \widetilde{\lhd}_p$). We define $\rho : \widetilde{E} \to Q$ to map an event $e \in \widetilde{E}$ to the control state that is associated with the sphere center of $\varsigma(\mathcal{S})$ where $r(e) = (\mathcal{S}, \nu)$ for some ν. In other words, let ρ be given by $\rho(e) = q$ if there are $\mathcal{S}$, ν, and σ such that $r(e) = (\mathcal{S}, \nu)$ and $\lambda(\mathcal{S}) = (\sigma, q)$. Then ρ turns out to be an accepting run of $\mathcal{B}$ on M. First, let $\widehat{\rho}$ be the mapping $\widetilde{E} \to \mathfrak{S}$ with $\widehat{\rho}(e) = H$ if there are $\mathcal{S}$ and ν such that $r(e) = (\mathcal{S}, \nu)$ and $H = \varsigma(\mathcal{S})$. For an extended R-sphere $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathfrak{S}^+$ and $e \in \widetilde{E}$, we often write $((E, \lhd, \lambda, \gamma, e_0), i) \in r(e)$ if there are $\mathcal{S}$ and ν such that $r(e) = (\mathcal{S}, \nu)$ and $((E, \lhd, \lambda, \gamma, e_0), i) \in \mathcal{S}$.

Claim 4.4.8 For each $e \in \widetilde{E}$, $((E, \lhd, \lambda, \gamma, \bar{e}), i) \in r(e)$, and $d \in \mathbb{N}$, if there is a sequence of events $e_0, \ldots, e_d \in E$ such that $e_0 = \bar{e}$ and, for each $k \in \{0, \ldots, d-1\}$, $e_k \lhd e_{k+1}$ or $e_{k+1} \lhd e_k$, then there is a unique sequence of events $\widehat{e}_0, \ldots, \widehat{e}_d \in \widetilde{E}$ with

- $\widehat{e}_0 = e$,
- for each $k \in \{0, \ldots, d\}$, $((E, \lhd, \lambda, \gamma, e_k), i) \in r(\widehat{e}_k)$, and
- for each $k \in \{0, \ldots, d-1\}$, $\widehat{e}_k \mathrel{\widetilde{\lhd}} \widehat{e}_{k+1}$ iff $e_k \lhd e_{k+1}$ and $\widehat{e}_{k+1} \mathrel{\widetilde{\lhd}} \widehat{e}_k$ iff $e_{k+1} \lhd e_k$.

Proof of Claim 4.4.8. We proceed by induction. Obviously, the statement holds for $d = 0$. Now assume there is a sequence of events $e_0, \ldots, e_d, e_{d+1} \in E$ such that $e_0 = \bar{e}$ and, for each $k \in \{0, \ldots, d\}$, $e_k \lhd e_{k+1}$ or $e_{k+1} \lhd e_k$. By induction hypothesis, there is a unique sequence of events $\widehat{e}_0, \ldots, \widehat{e}_d \in \widetilde{E}$ with

- $\widehat{e}_0 = e$,
- for each $k \in \{0, \ldots, d\}$, $((E, \lhd, \lambda, \gamma, e_k), i) \in r(\widehat{e}_k)$ (in particular, $\lambda(e_k) = (\widetilde{\lambda}(\widehat{e}_k), q)$ for some $q \in Q$), and

- for each $k \in \{0, \ldots, d-1\}$, $\widehat{e}_k \mathrel{\widetilde{\lhd}} \widehat{e}_{k+1}$ iff $e_k \lhd e_{k+1}$ (which implies, for one thing, $\widehat{e}_k \mathrel{\widetilde{\lhd}_{\mathrm{c}}} \widehat{e}_{k+1}$ iff $e_k \lhd_{\mathrm{c}} e_{k+1}$) and $\widehat{e}_{k+1} \mathrel{\widetilde{\lhd}} \widehat{e}_k$ iff $e_{k+1} \lhd e_k$.

Suppose that

- $e_d \lhd_p e_{d+1}$ for some $p \in P$. As e_d is not maximal in $(E_p, <_p)$, $r(\widehat{e}_d)$ cannot be part of a final state so that there is a (unique) event $\widehat{e}_{d+1} \in \widetilde{E}$ with $\widehat{e}_d \mathrel{\widetilde{\lhd}_p} \widehat{e}_{d+1}$. Furthermore, due to item 6. from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e_{d+1}), i) \in r(\widehat{e}_{d+1})$.

- $e_{d+1} \lhd_p e_d$ for some $p \in P$. As e_d is not minimal in $(E_p, <_p)$, there is, according to item 5. from the definition of Δ_p, a (unique) event $\widehat{e}_{d+1} \in \widetilde{E}$ with $\widehat{e}_{d+1} \mathrel{\widetilde{\lhd}_p} \widehat{e}_d$ and $((E, \lhd, \lambda, \gamma, e_{d+1}), i) \in r(\widehat{e}_{d+1})$.

- $e_d \lhd_{\mathrm{c}} e_{d+1}$. There is a (unique) event $\widehat{e}_{d+1} \in \widetilde{E}$ with $\widehat{e}_d \mathrel{\widetilde{\lhd}_{\mathrm{c}}} \widehat{e}_{d+1}$. Set $(\mathcal{P}, \mathcal{N})$ to be $m((\widehat{e}_d, \widehat{e}_{d+1}))$. According to item 7. (i) (a) from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e_{d+1}), i) \in \mathcal{P}$. With 7. (ii) (a), it follows $((E, \lhd, \lambda, \gamma, e_{d+1}), i) \in r(\widehat{e}_{d+1})$.

- $e_{d+1} \lhd_{\mathrm{c}} e_d$. There is a (unique) event $\widehat{e}_{d+1} \in \widetilde{E}$ with $\widehat{e}_{d+1} \mathrel{\widetilde{\lhd}_{\mathrm{c}}} \widehat{e}_d$. Set $(\mathcal{P}, \mathcal{N})$ to be $m((\widehat{e}_{d+1}, \widehat{e}_d))$. According to item 7. (ii) (c) from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e_d), i) \in \mathcal{P}$. With 7. (i) (c), it follows $((E, \lhd, \lambda, \gamma, e_{d+1}), i) \in r(\widehat{e}_{d+1})$.

This concludes the proof of Claim 4.4.8. ■

We have to show that, for each $e \in \widetilde{E}$, the R-sphere of $(\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_{\mathrm{c}}, (\widetilde{\lambda}, \rho))$ around e is isomorphic to $\widehat{\rho}(e)$. So let $e \in \widetilde{E}$ and set $(E, \lhd, \lambda, \gamma)$ to be $\widehat{\rho}(e)$ and $i \in \{1, \ldots, 4 \cdot maxE^2 + 1\}$ to be the unique element with $((E, \lhd, \lambda, \gamma, \gamma), i) \in r(e)$.

Claim 4.4.9 For each $d \in \{0, \ldots, R\}$, there is an isomorphism

$$h : d\text{-Sph}((\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_{\mathrm{c}}, (\widetilde{\lambda}, \rho)), e) \rightarrow d\text{-Sph}((E, \lhd, \lambda), \gamma)$$

such that, for each $\widehat{e} \in \widetilde{E}$ with $d(\widehat{e}, e) \leq d$, $((E, \lhd, \lambda, \gamma, h(\widehat{e})), i) \in r(\widehat{e})$.

Proof of Claim 4.4.9. We proceed by induction. The statement holds for $d = 0$. Now assume $d < R$ and there is an isomorphism $h : d\text{-Sph}((\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_{\mathrm{c}}, (\widetilde{\lambda}, \rho)), e) \rightarrow d\text{-Sph}((E, \lhd, \lambda), \gamma)$ such that, for each $\widehat{e} \in \widetilde{E}$ with $d(\widehat{e}, e) \leq d$, $((E, \lhd, \lambda, \gamma, h(\widehat{e})), i) \in r(\widehat{e})$.

Extended sphere simulates MSC Suppose there is $\widehat{e}_1, \widehat{e}'_1, \widehat{e}_2, \widehat{e}'_2 \in \widetilde{E}$ such that $d(\widehat{e}_1, e) = d(\widehat{e}_2, e) = d$, $d(\widehat{e}'_1, e) = d(\widehat{e}'_2, e) = d + 1$, ($\widehat{e}_1 \mathrel{\widetilde{\lhd}} \widehat{e}'_1$ or $\widehat{e}'_1 \mathrel{\widetilde{\lhd}} \widehat{e}_1$), and ($\widehat{e}_2 \mathrel{\widetilde{\lhd}} \widehat{e}'_2$ or $\widehat{e}'_2 \mathrel{\widetilde{\lhd}} \widehat{e}_2$). Furthermore, suppose (let e_1 and e_2 denote $h(\widehat{e}_1)$ and $h(\widehat{e}_2)$, respectively)

- $\widehat{e}_1 \mathrel{\widetilde{\lhd}_p} \widehat{e}'_1$ for some $p \in P$. As $d(\widehat{e}_1, e) < R$, we have $d(e_1, \gamma) < R$. Due to item 4. from the definition of Δ_p, e_1 is not maximal in $(E_p, <_p)$ so that there is $e'_1 \in E$ with $e_1 \lhd_p e'_1$ and, due to item 6. and $((E, \lhd, \lambda, \gamma, e_1), i) \in r(\widehat{e}_1)$, $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$.

- $\widehat{e}'_1 \mathrel{\widetilde{\lhd}_p} \widehat{e}_1$ for some $p \in P$. As $d(\widehat{e}_1, e)$ is less than R, so is $d(e_1, \gamma)$. Due to item 3. from the definition of Δ_p, e_1 is not minimal in $(E_p, <_p)$ so that there is $e'_1 \in E$ with $e'_1 \lhd_p e_1$ and, due to item 5. and $((E, \lhd, \lambda, \gamma, e_1), i) \in r(\widehat{e}_1)$, $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$.

- $\widehat{e}_1 \mathrel{\widetilde{\lhd}_c} \widehat{e}'_1$. Set $(\mathcal{P}, \mathcal{N})$ to be $m((\widehat{e}_1, \widehat{e}'_1))$. As $d(\widehat{e}_1, e) < R$ and, thus, $d(e_1, \gamma) < R$, there is $e'_1 \in E$ such that $e_1 \lhd_c e'_1$. (This is because $(E, \lhd, \lambda, \gamma)$ can be embedded into some MSC.) According to item 7. (i) (a) from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e'_1), i) \in \mathcal{P}$. Due to item 7. (ii) (a), it then follows from $((E, \lhd, \lambda, \gamma, e_1), i) \in r(\widehat{e}_1)$ that $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$.

- $\widehat{e}'_1 \mathrel{\widetilde{\lhd}_c} \widehat{e}_1$. Set $(\mathcal{P}, \mathcal{N})$ to be $m((\widehat{e}'_1, \widehat{e}_1))$. As $d(\widehat{e}_1, e) < R$ and, consequently, $d(e_1, \gamma) < R$, there is also $e'_1 \in E$ such that $e'_1 \lhd_c e_1$. (Recall that $(E, \lhd, \lambda, \gamma)$ can be embedded into some MSC.) According to item 7. (ii) (c) from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e_1), i) \in \mathcal{P}$. Due to item 7. (i) (c), it then follows that $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$.

Thus, depending on $\widehat{e}'_1$, we obtain from e_1 a unique event $e'_1 \in E$, which we denote by $h'(\widehat{e}'_1)$. According to the above scheme, we obtain from e_2 a unique event $e'_2 \in E$, denoted by $h'(\widehat{e}'_2)$. It holds $d(e'_1, \gamma) = d(e'_2, \gamma) = d+1$. Now suppose

- $\widehat{e}'_1 \mathrel{\widetilde{\lhd}_p} \widehat{e}'_2$ for some $p \in P$. As we already have $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$ and $((E, \lhd, \lambda, \gamma, e'_2), i) \in r(\widehat{e}'_2)$, it follows from item 2. of the definition of Δ_p that $e'_1 \lhd_p e'_2$.

- $\widehat{e}'_1 \mathrel{\widetilde{\lhd}_c} \widehat{e}'_2$. Set $(\mathcal{P}, \mathcal{N})$ to be $m((\widehat{e}'_1, \widehat{e}'_2))$ and suppose $e'_1 \lhd_c e'_2$ does not hold. But then, according to items 7. (i) (b) and 7. (ii) (b) from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e'_2), i) \in \mathcal{N}$ and $((E, \lhd, \lambda, \gamma, e'_2), i) \notin r(\widehat{e}'_2)$, resulting in a contradiction.

- $\widehat{e}'_1 = \widehat{e}'_2$. Then $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$ and $((E, \lhd, \lambda, \gamma, e'_2), i) \in r(\widehat{e}'_1)$ implies $e'_1 = e'_2$ (otherwise, $r(\widehat{e}'_1)$ would not be a valid state of $\mathcal{A}$).

The cases $\widehat{e}'_2 \mathrel{\widetilde{\lhd}_p} \widehat{e}'_1$ and $\widehat{e}'_2 \mathrel{\widetilde{\lhd}_c} \widehat{e}'_1$ are handled analogously.

MSC simulates extended sphere Suppose there is $e_1, e'_1, e_2, e'_2 \in E$ such that $d(e_1,\gamma) = d(e_2,\gamma) = d$, $d(e'_1,\gamma) = d(e'_2,\gamma) = d+1$, ($e_1 \lhd e'_1$ or $e'_1 \lhd e_1$) and ($e_2 \lhd e'_2$ or $e'_2 \lhd e_2$). We now proceed as in the proof of Claim 4.4.8. So suppose (let $\widehat{e}_1$ and $\widehat{e}_2$ denote $h^{-1}(e_1)$ and $h^{-1}(e_2)$, respectively)

- $e_1 \lhd_p e'_1$ for some $p \in P$. As e_1 is not maximal in $(E_p, <_p)$, $r(\widehat{e}_1)$ cannot be part of a final state so that there is $\widehat{e}'_1 \in \widetilde{E}$ with $\widehat{e}_1 \mathrel{\widetilde{\lhd}_p} \widehat{e}'_1$. Furthermore, due to item 6. from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$.
- $e'_1 \lhd_p e_1$ for some $p \in P$. As e_1 is not minimal in $(E_p, <_p)$ there is, according to item 5. from the definition of Δ_p, $\widehat{e}'_1 \in \widetilde{E}$ with $\widehat{e}'_1 \mathrel{\widetilde{\lhd}_p} \widehat{e}_1$ and $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$.
- $e_1 \lhd_c e'_1$. There is $\widehat{e}'_1 \in \widetilde{E}$ with $\widehat{e}_1 \mathrel{\widetilde{\lhd}_c} \widehat{e}'_1$. Set $(\mathcal{P}, \mathcal{N})$ to be $m((\widehat{e}_1, \widehat{e}'_1))$. According to item 7. (i) (a) from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e'_1), i) \in \mathcal{P}$. With 7. (ii) (a), it follows $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$.
- $e'_1 \lhd_c e_1$. There is $\widehat{e}'_1 \in \widetilde{E}$ with $\widehat{e}'_1 \mathrel{\widetilde{\lhd}_c} \widehat{e}_1$. Set $(\mathcal{P}, \mathcal{N})$ to be $m((\widehat{e}'_1, \widehat{e}_1))$. According to item 7. (ii) (c) from the definition of Δ_p, $((E, \lhd, \lambda, \gamma, e_1), i) \in \mathcal{P}$. With 7. (i) (c), it follows $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$.

According to the above scheme, we obtain from $\widehat{e}_2$ a unique event $\widehat{e}'_2$. Now suppose

- $e'_1 \lhd_p e'_2$ for some $p \in P$. Assume $\widehat{e}'_1 \mathrel{\widetilde{\not\lhd}_p} \widehat{e}'_2$. According to the definition of the set of states of $\mathcal{A}$, $e'_1 \neq e'_2$, $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$, and $((E, \lhd, \lambda, \gamma, e'_2), i) \in r(\widehat{e}'_2)$ implies $\widehat{e}'_1 \neq \widehat{e}'_2$. But then, following the scheme depicted in Figure 4.8, we can construct an infinite sequence $x_1, x_2, \ldots \in \widetilde{E}$ inducing an infinite set of (pairwise distinct) events: Suppose $\widehat{e}'_1 \mathrel{\widetilde{<}_p} \widehat{e}'_2$. (The other case is handled analogously.) Set $x_1 \in \widetilde{E}$ to be the unique event satisfying $\widehat{e}'_1 \mathrel{\widetilde{\lhd}_p} x_1$. We have $((E, \lhd, \lambda, \gamma, e'_2), i) \in r(x_1)$ and $x_1 \mathrel{\widetilde{<}_p} \widehat{e}'_2$. According to Claim 4.4.8, there is $x_2 \in \widetilde{E}$ such that $((E, \lhd, \lambda, \gamma, \gamma), i) \in r(x_2)$ and $x_2 \mathrel{\widetilde{<}_{P(e)}} e$. (There is a path in $(E, \lhd, \lambda)$ from e'_2 to γ that, according to Claim 4.4.8, takes M from $\widehat{e}'_2$ to e. Apply this path to x_1 yielding a path to a unique event $x_2 \in \widetilde{E}$ with $((E, \lhd, \lambda, \gamma, \gamma), i) \in r(x_2)$. From $x_1 \mathrel{\widetilde{<}_p} \widehat{e}'_2$, it easily follows that $x_2 \mathrel{\widetilde{<}_{P(e)}} e$.) Similarly, there is $x_3 \in \widetilde{E}$ with $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(x_3)$ and $x_3 \mathrel{\widetilde{<}_p} \widehat{e}'_1$. Now let $x_4 \in \widetilde{E}$ be the unique event such that $x_3 \mathrel{\widetilde{\lhd}_p} x_4$ and $((E, \lhd, \lambda, \gamma, e'_2), i) \in r(x_4)$ (as already $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(\widehat{e}'_1)$, it holds $x_4 \mathrel{\widetilde{<}_p} \widehat{e}'_1$) and let, again following Claim 4.4.8, $x_5 \in \widetilde{E}$ be an event with $((E, \lhd, \lambda, \gamma, \gamma), i) \in r(x_5)$ and $x_5 \mathrel{\widetilde{<}_{P(x_2)}} x_2$ and $x_6 \in \widetilde{E}$ be an event with $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(x_6)$ and $x_6 \mathrel{\widetilde{<}_p} x_3$. Continuing this scheme yields an infinite set of events, contradicting the premise that we deal with finite MSCs.

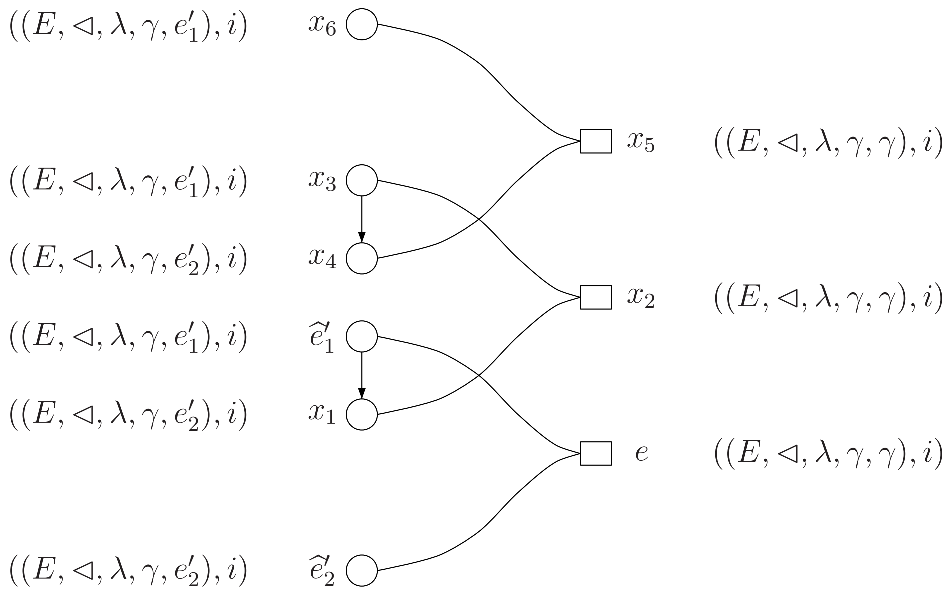


Figure 4.8: An infinite sequence of events

- $e'_1 \lhd_{\mathrm{c}} e'_2$. Assuming $\widehat{e}'_1 \not\mathrel{\widetilde{\lhd}}_{\mathrm{c}} \widehat{e}'_2$, we proceed according to the very same scheme as in case $e'_1 \lhd_p e'_2$ to generate an infinite sequence $x_1, x_2, \ldots \in \widetilde{E}$ inducing an infinite set of events, i.e., set $x_1 \in \widetilde{E}$ to be the unique event such that $\widehat{e}'_1 \mathrel{\widetilde{\lhd}}_{\mathrm{c}} x_1$ and $((E, \lhd, \lambda, \gamma, e'_2), i) \in r(x_1)$. Assuming $x_1 \mathrel{\widetilde{<}}_{P(\widehat{e}'_2)} \widehat{e}'_2$, we can find $x_2 \in \widetilde{E}$ with $((E, \lhd, \lambda, \gamma, \gamma), i) \in r(x_2)$ and $x_2 \mathrel{\widetilde{<}}_{P(e)} e$ and so on.

- $e'_1 = e'_2$. Again, assuming $\widehat{e}'_1 \neq \widehat{e}'_2$, we generate a sequence $x_1, x_2, \ldots \in \widetilde{E}$ inducing an infinite set of events as follows: Suppose $\widehat{e}'_1 \mathrel{\widetilde{<}}_{P(\widehat{e}'_2)} \widehat{e}'_2$. According to Claim 4.4.8, we can find $x_1 \in \widetilde{E}$ such that $((E, \lhd, \lambda, \gamma, \gamma), i) \in r(x_1)$ and $x_1 \mathrel{\widetilde{<}}_{P(e)} e$. Furthermore, there is $x_2 \in \widetilde{E}$ satisfying $((E, \lhd, \lambda, \gamma, e'_1), i) \in r(x_2)$ and $x_2 \mathrel{\widetilde{<}}_{P(\widehat{e}'_1)} \widehat{e}'_1$ and so on.

The cases $e'_2 \lhd_p e'_1$ and $e'_2 \lhd_{\mathrm{c}} e'_1$ are handled analogously. From the above results, we can conclude that the mapping $\widehat{h} : (d+1)\text{-Sph}((\widetilde{E}, \{\widetilde{\lhd}_p\}_{p \in P}, \widetilde{\lhd}_{\mathrm{c}}, (\widetilde{\lambda}, \rho)), e) \to (d+1)\text{-Sph}((E, \lhd, \lambda), \gamma)$ given by

$$\widehat{h}(\widehat{e}) = \begin{cases} h(\widehat{e}) & \text{if } d(\widehat{e}, e) \leq d \\ h'(\widehat{e}) & \text{if } d(\widehat{e}, e) = d+1 \end{cases}$$

(for $\widehat{e} \in \widetilde{E}$ with $d(\widehat{e}, e) \leq d+1$) is an isomorphism satisfying, for any $\widehat{e} \in \widetilde{E}$ with $d(\widehat{e}, e) \leq d+1$, $((E, \lhd, \lambda, \gamma, \widehat{h}(\widehat{e})), i) \in r(\widehat{e})$. This concludes the proof of Claim 4.4.9. ■

As $((\mathcal{S}_p, \nu_p))_{p \in P} \in F$ only if the union of mappings ν_p is a model of Occ, an accepting run of $\mathcal{A}$ makes sure that the number of occurrences of an R-sphere meets the obligations imposed by $\mathcal{B}$. This concludes the proof of Claim 4.4.7 and the proof of Lemma 4.4.4. $\square$

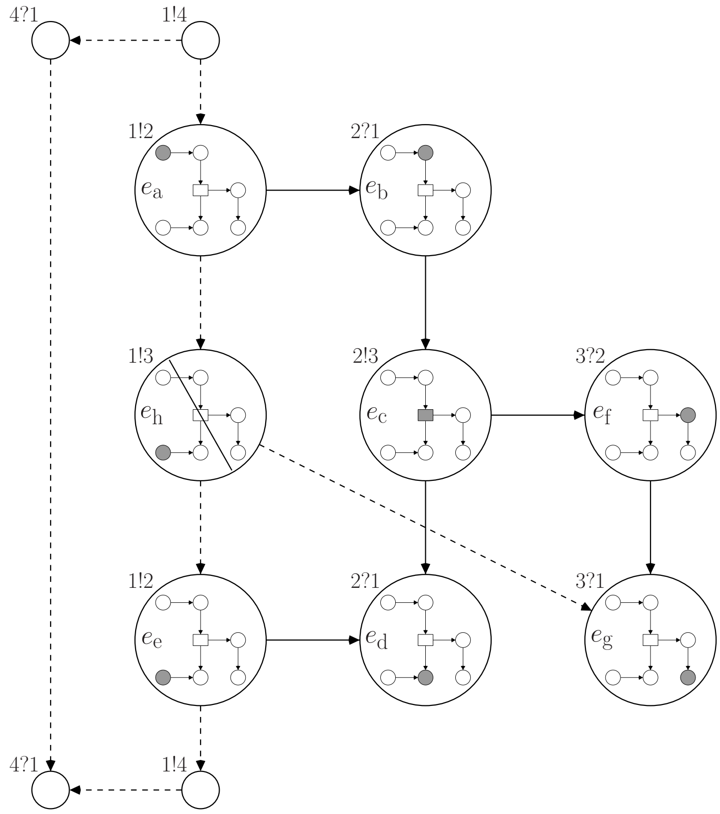


Figure 4.9: Simulating a graph acceptor

Example 4.4.10 In the following, let H denote the 2-sphere in part (a) from Figure 4.4 on page 75. Figure 4.9, showing some MSC M with four processes, illustrates the transition behavior of the MPA $\mathcal{A}$. It demonstrates how a run of $\mathcal{A}$ on M transfers extensions of H from one event of M to a neighboring one to make sure that the 2-sphere around event e_c (which is indicated by solid edges) is isomorphic to H. For example, the state that is taken on event e_a may contain the extended sphere (H, a). (For clarity, control states and the natural i to distinguish different instances of spheres are omitted.) As a $\lhd_c$ b (wrt. the edge relation of H), $\mathcal{A}$ passes (H, b) in form of a message to process 2. Receiving (H, b), process 2 becomes aware it should bind e_b to some state that contains (H, b) (Conditions 7. (i) (a) and 7. (ii) (a) from the definition of the transition relation). As, in H, b is followed by c, so e_c has to be associated with a state containing (H, c) (Condition

6.). In contrast, e_{h} is not allowed to carry the extended sphere (H, e), unless it belongs to a different instance of H (Condition 2.). Now consider e_{d}, which holds the extended sphere (H, d). Due to Condition 5., the preceding state, which is associated to e_{c}, must contain (H, c), which means that a run cannot simply enter H beginning with d. Moreover, as e_{d} is a receive event, $\mathcal{A}$ has to receive a message containing (H, d) (Condition 7. (ii) (c)). In turn, the corresponding send event e_{e} has to be associated with a state that holds (H, e) (Condition 7. (i) (c)). Note that, as $d(a, c) = d(e, c) = 2$, the (illustrated parts of the) states assigned to e_{a} and e_{e} satisfy Conditions 3. and 4.

4.4.4 1-Spheres suffice

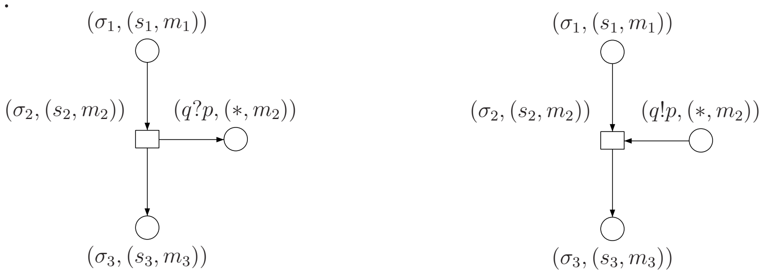
Based on the results from the last section, we can now conclude that already graph acceptors with radius 1 suffice to cover $\mathcal{EMSO}_{\mathbb{MSC}}$. In the context of pictures, a corresponding reduction has been applied to tiling systems [GRST96].

Corollary 4.4.11

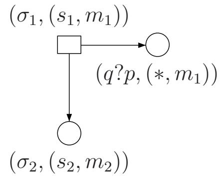
$$1\text{-}\mathcal{GA}_{\mathbb{MSC}} = \mathcal{GA}_{\mathbb{MSC}}$$

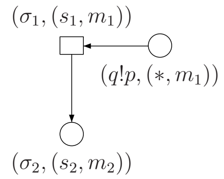
Proof Let $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, be a finite MPA. A corresponding graph acceptor $\mathcal{B} = (Q, R, \mathfrak{S}, Occ)$ over (Act, P_{c}) with $R = 1$ and $L_{\mathbb{MSC}}(\mathcal{B}) = L(\mathcal{A})$ is given as follows:

- $Q = \bigcup_{p \in P}(S_p \times \mathcal{D})$
- for any $p, q \in P$, $s_0, s_1, s_2, s_3 \in S_p$, and $(\sigma_1, m_1), (\sigma_2, m_2), (\sigma_3, m_3) \in Act_p \times \mathcal{D}$ with $(s_i, (\sigma_{i+1}, m_{i+1}), s_{i+1}) \in \Delta_p$ $(i = 0, 1, 2)$, $\mathfrak{S}$ contains (for any $* \in S_q$) the left-hand side of the following if $\sigma_2 = p!q$ and the right-hand side if $\sigma_2 = p?q$:

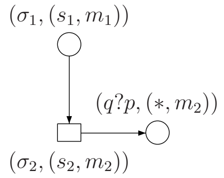


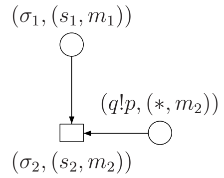
- for any $p, q \in P$, $s_1, s_2 \in S_p$, and $(\sigma_1, m_1), (\sigma_2, m_2) \in Act_p \times \mathcal{D}$ with $(\overline{s}^{in}[p], (\sigma_1, m_1), s_1) \in \Delta_p$ and $(s_1, (\sigma_2, m_2), s_2) \in \Delta_p$, $\mathfrak{S}$ contains (for any $* \in S_q$) the left-hand side of the following if $\sigma_1 = p!q$ and the right-hand side if $\sigma_1 = p?q$:



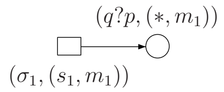


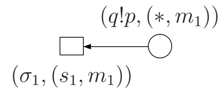
- for any $p, q \in P$, $s_0, s_1, s_2 \in S_p$, and $(\sigma_1, m_1), (\sigma_2, m_2) \in Act_p \times \mathcal{D}$ with $(s_0, (\sigma_1, m_1), s_1) \in \Delta_p$ and $(s_1, (\sigma_2, m_2), s_2) \in \Delta_p$, $\mathfrak{S}$ contains (for any $* \in S_q$) the left-hand side of the following if $\sigma_2 = p!q$ and the right-hand side if $\sigma_2 = p?q$:





- for any $p, q \in P$, $s_1 \in S_p$, and $(\sigma_1, m_1) \in Act_p \times \mathcal{D}$ with $(\overline{s}^{in}[p], (\sigma_1, m_1), s_1) \in \Delta_p$, $\mathfrak{S}$ contains (for any $* \in S_q$) the left-hand side of the following if $\sigma_1 = p!q$ and the right-hand side if $\sigma_1 = p?q$:





- $Occ = \bigvee_{\overline{s} \in F} \bigwedge_{p \in P} \bigvee_{H \in FT_{p,\overline{s}[p]}}$ "$H \geq 1$"
  where, for $p \in P$ and $s \in S_p$, $FT_{p,s}$ shall contain exactly those spheres whose center is both maximal on the process line of p and labeled with s.$\square$

In turn, the construction of an MPA from a graph acceptor with 1-spheres is a lot easier. So let $\mathcal{B} = (Q, R, \mathfrak{S}, Occ)$ be a graph acceptor over (Act, P_c) with $R = 1$. Again, it suffices to consider only those R-spheres $H \in \mathfrak{S}$ for which there is an extended MSC $M = (E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda) \in \mathbb{MSC}^Q$, which has an extended labeling function $\lambda : E \to Act \times Q$, and an event $e \in E$ such that H is the R-sphere of M *around* e. In particular, any sphere center γ can refer to a node that is part of the same sphere and that can be interpreted as the corresponding communication event.

Let us recall and introduce some notions. For $p \in P$, let $\mathfrak{S}_p$ denote the set of spheres $(E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda, \gamma) \in \mathfrak{S}$ such that $\gamma \in E_p$. Similarly, for $\sigma \in Act$, $\mathfrak{S}_\sigma$ is the set of spheres whose sphere center is labeled with σ (together with some state from Q). Moreover, $\mathfrak{S}_{min}/\mathfrak{S}_{max}$ denotes the set of all those spheres $(E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda, \gamma) \in \mathfrak{S}$ where γ is minimal/maximal in $(E_{P(\gamma)}, \leq_{P(\gamma)})$, respectively. Finally, let $\max(Occ)$ denote the least threshold n such that Occ does not distinguish occurrence numbers $\geq n$.

Now, $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, can be specified as follows: For $p \in P$, $S_p = (\{\iota_p\} \uplus \mathfrak{S}_p) \times \{\nu \mid \nu : \mathfrak{S}_p \to \{0, \ldots, \max(Occ)\}\}$, i.e., a p-local state of $\mathcal{A}$ is a pair whose first component is either ι_p, which will be the p-local initial state, or an R-sphere from $\mathfrak{S}_p$ and whose second component is a mapping that keeps track of the number of spheres used so far. Recall that, for $p \in P$, ν_p^0 denotes the initial mapping, which maps each R-sphere $H \in \mathfrak{S}_p$ to 0, and, for $\nu : \mathfrak{S}_p \to \{0, \ldots, \max(Occ)\}$, $H \in \mathfrak{S}_p$, and a natural $n \in \{0, \ldots, \max(Occ)\}$, $\nu[H/n]$ denotes the assignment that maps H to n and, otherwise, coincides with ν. The set $\mathcal{D}$ of synchronization messages is the set $\mathfrak{S}$ of R-spheres. A transition from Δ_p has one of the following types (transitions from local initial states require to be either of type (IS) or (IR), while, afterwards, only (TS)- and (TR)-transitions can be applied):

(TS) There is a transition $((H, \nu), p!q, G, (H', \nu')) \in \Delta_p$ if there exists an extended MSC $M = (E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda) \in \mathbb{MSC}^Q$ (recall that $\lambda : E \to Act \times Q$) and events $e_1, e \in E$ such that

1. $e_1 \lhd_p e$,
2. H is the R-sphere of M around e_1,
3. $H' = G \in \mathfrak{S}_{p!q}$ is the R-sphere of M around e, and
4. $\nu' = \nu[H' / \min\{\nu(H') + 1, \max(Occ)\}]$.

(TR) There is a transition $((H, \nu), p?q, G, (H', \nu')) \in \Delta_p$ if there exists an extended MSC $M = (E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda)$ and events $e_1, e_2, e \in E$ such that

1. $e_1 \lhd_p e$ and $e_2 \lhd_c e$,
2. H is the R-sphere of M around e_1,
3. G is the R-sphere of M around e_2,
4. $H' \in \mathfrak{S}_{p?q}$ is the R-sphere of M around e, and
5. $\nu' = \nu[H' / \min\{\nu(H') + 1, \max(Occ)\}]$.

(IS) There is a transition $((\iota_p, \nu_p^0), p!q, G, (H', \nu')) \in \Delta_p$ if

1. $H' = G \in \mathfrak{S}_{p!q} \cap \mathfrak{S}_{min}$ and
2. $\nu' = \nu_p^0[H' / \min\{1, \max(Occ)\}]$.

(IR) There is a transition $((\iota_p, \nu_p^0), p?q, G, (H', \nu')) \in \Delta_p$ if there exists an extended MSC $M = (E, \{\lhd_p\}_{p\in P}, \lhd_c, \lambda)$ and events $e_1, e \in E$ such that

1. $e_1 \lhd_c e$,
2. G is the R-sphere of M around e_1,
3. $H' \in \mathfrak{S}_{p?q} \cap \mathfrak{S}_{min}$ is the R-sphere of M around e, and
4. $\nu' = \nu_p^0[H'/\min\{1, \max(Occ)\}]$.

The initial state $\overline{s}^{in}$ is given by $((\iota_p, \nu_p^0))_{p\in P}$, while a global state of $\mathcal{A}$, which, for each $p \in P$, is supposed to employ an element $s_p \in \{\iota_p\} \uplus \mathfrak{S}_p$ and a mapping ν_p, is contained in F if $\bigcup_{p\in P} \nu_p$ (with the expected meaning) satisfies the requirements imposed by Occ and, for all $p \in P$, $s_p \in \{\iota_p\} \cup \mathfrak{S}_{max}$.

Claim 4.4.12 $L(\mathcal{A}) = L_{\mathbb{MSC}}(\mathcal{B})$

Proof of Claim 4.4.12. Let $\rho : E \to Q$ be an accepting run of $\mathcal{B}$ on the MSC $M = (E, \{\lhd_p\}_{p\in P}, \lhd_c, \lambda)$ and let in the following $\widehat{\rho}$ denote the mapping $E \to \mathfrak{S}$ that maps an event $e \in E$ onto the sphere of $(E, \{\lhd_p\}_{p\in P}, \lhd_c, (\lambda, \rho))$ around e. For $H \in \mathfrak{S}$ and $e \in E$, let furthermore $le_{M,\rho}(H, e) := |\{e' \in E_{P(e)} \mid e' \leq_{P(e)} e,\ H \cong \widehat{\rho}(e')\}|$. We then define $r : E \to \bigcup_{p\in P} S_p$ to map an event $e \in E$ onto $(\widehat{\rho}(e), \nu)$ where, for $H \in \mathfrak{S}_{P(e)}$, $\nu(H) = \min\{le_{M,\rho}(H, e), \max(Occ)\}$. Finally, let $m : \lhd_c \to \mathcal{D}$ map a pair $(e_s, e_r) \in \lhd_c$ onto $\widehat{\rho}(e_s)$. We easily verify that (r, m) is an accepting run of $\mathcal{A}$ on M.

If, conversely, (r, m) is an accepting run of $\mathcal{A}$ on $M = (E, \{\lhd_p\}_{p\in P}, \lhd_c, \lambda) \in \mathbb{MSC}$ and $\rho : E \to Q$ maps an event $e \in E$ to the control state that is associated with the sphere center of $r(e)$, then ρ is an accepting run of $\mathcal{B}$ on M. $\square$

One might be tempted to apply the above transformation from special graph acceptors with radius 1 into MPAs to graph acceptors with arbitrarily large radius. However, this attempt fails. For example, consider the MSC M from Figure 4.10 and a graph acceptor $\mathcal{B}$ without occurrence constraints and a singleton as a state space, which, among others, is equipped with the 4-sphere H_a from Figure 4.11, which is the 4-sphere of M around A but with missing edge between E and F, and furthermore contains the 4-sphere of M around B but with missing edge between F and G, and so on (cf. Figures A.1–A.10 on pages 137–145 for formal definitions). Now suppose $\mathcal{B}$ runs on M. In fact, any of the transitions (edges) between events from $\{A, B, C, D, E, F, G, H, I\}$ is witnessed by some embedding into an (extended) MSC. More precisely, in a successful run on M, we may assign to A sphere H_a, to B sphere H_b and so on. Though the 4-sphere of M around

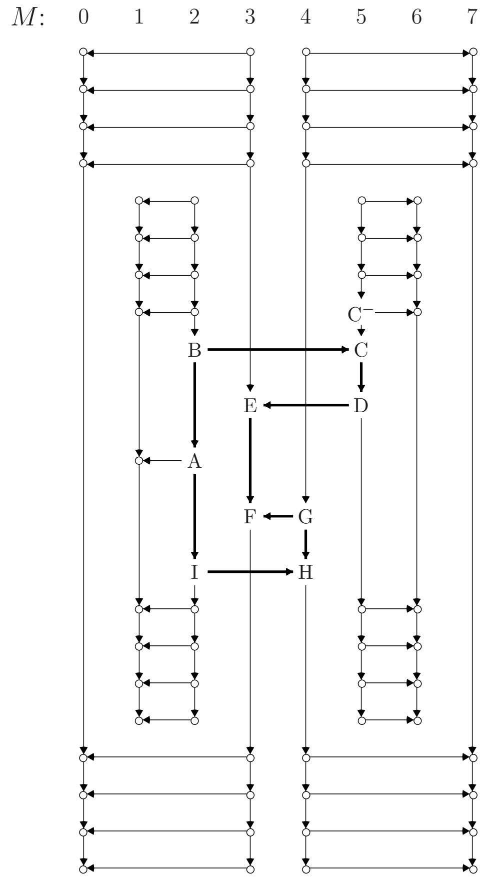


Figure 4.10: Counterexample for the simpler transformation

A is not isomorphic to some sphere from $\mathcal{B}$ (in particular, it is not isomorphic to the state assigned to A), the run is accepting. The reason is that two neighboring transitions can be justified by two embeddings into completely different MSCs so that the graph acceptor possibly forgets that, actually, there has to be an event between, say, E and F to ensure isomorphism between a sphere and the MSC. In other words, it is possible to "carry around" the missing edge without the need to actually ensure that there is an event in between instead of just one edge.

Let us discuss one transition in more detail. In the sense of the definition from page 91, the transition $(H_{\mathrm{e}}, 3?4, H_{\mathrm{g}}, H_{\mathrm{f}})$ of type (TR) (recall that we disregard occurrence constraints) is a valid one, as witnessed by the MSC M_{egf} from Figure A.5 on page 140. In M_{egf}, the 4-sphere around E is isomorphic to H_{e}, the 4-sphere around F is isomorphic to H_{f}, and the 4-sphere around G is isomorphic to H_{g}. See the appendix for further transitions.

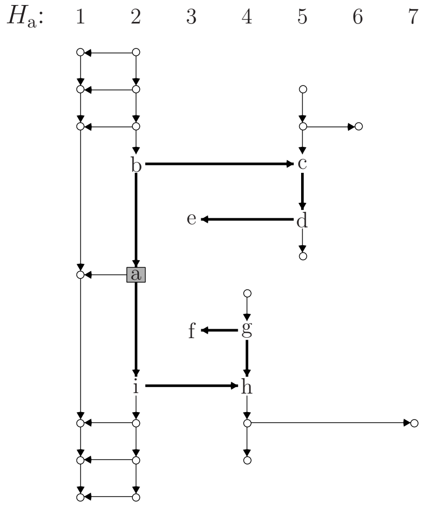


Figure 4.11: Counterexample for the simpler transformation

4.4.5 MPAs vs. Graph Acceptors

In the last sections, we have implicitly shown that $\mathcal{MPA}$ and $\mathcal{GA}_{\mathbb{MSC}}$ coincide. Let us address graph acceptors over MSCs a little longer and carry over some results concerning traces and product trace languages. As in most settings, we cannot generally remove occurrence constraints from graph acceptors over MSCs. But let us turn our attention to Lemma 2.4.13, which is adapted as follows:

Lemma 4.4.13 For any $L \in \mathcal{L}(\mathrm{MPA}_\ell)$, we have

$$L \in \mathcal{GA}^{-}_{\mathbb{MSC}} \text{ iff } L \in \mathcal{GA}_{\mathbb{MSC}}.$$

Proof We start from the construction of the proof of Corollary 4.4.11, where, for a finite MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, an equivalent graph acceptor $\mathcal{B} = (Q, R, \mathfrak{S}, Occ)$ over (Act, P_c) with $R = 1$ and $L_{\mathbb{MSC}}(\mathcal{B}) = L(\mathcal{A})$ is built. However, if $\mathcal{A}$ is locally accepting, F is the cartesian product of sets $F_p \subseteq S_p$. Thus, Occ reduces to

$$\bigwedge_{p \in P} \left(\bigvee_{s \in F_p, H \in FT_{p,s}} \text{“}H \geq 1\text{”} \right),$$

which can be assumed to be true if we remove any sphere from $\mathfrak{S}$ that, wrt. some process $p \in P$, has a maximal sphere center that is not labeled with a state from F_p and therefore does not belong to some $FT_{p,s}$ with $s \in F_p$. $\square$

Note that $\mathcal{EP}^0 \subseteq \mathcal{L}(\mathrm{MPA}^f_\ell)$ so that, in particular, the above result holds for weak EMSO-definable product MSC languages. Altogether, we observe the following correspondence: while the role of messages in an MPA corresponds to the role of control states of graph acceptors, final states correspond to occurrence constraints. The class $\mathcal{L}(\mathrm{MPA}^f_\ell)$ is not only interesting because of the above property. It is the basis for an algorithm by Genest et al. that, given a locally-cooperating high-level MSC, yields a corresponding locally-accepting finite MPA [GMSZ02]. Furthermore, it turns out that, if we restrict to connected MSCs, $\mathcal{L}(\mathrm{MPA}^f_\ell)$ and $\mathcal{EMSO}_{\mathbb{MSC}}$ coincide.

Lemma 4.4.14 Let L be an MSC language. If, for any $M \in L$, M is connected, then

$$L \in \mathcal{L}(\mathrm{MPA}^f_\ell) \;\text{ iff }\; L \in \mathcal{L}(\mathrm{MPA}^f).$$

Proof Let $\mathcal{A} = ((\mathcal{A}_p)_{p\in P}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, be a finite MPA such that, for any $M \in L(\mathcal{A})$, $\mathrm{c}G(M)$ is connected. We are interested in a locally-accepting finite MPA $\mathcal{A}' = ((\mathcal{A}'_p)_{p\in P}, \mathcal{D}', \overline{s}'_0, F')$, $\mathcal{A}_p = (S'_p, \Delta'_p)$, such that $L(\mathcal{A}') = L(\mathcal{A})$. Let us first describe the idea behind the construction of $\mathcal{A}'$. Each process, when executing its first transition, will basically guess both a global final state and a connected communication graph. In addition, a component p of a global final state might be $-$, which shall indicate that p is not expected to move at all. In that case, p is not a node of the communication graph. Once a global final state and a communication graph are guessed, they cannot be changed anymore. Furthermore, they are passed to communication partners, which, in turn, are only allowed to reply the request if they have made the same choice. Simultaneously, a set of previous communication partners is locally updated and, at the end of a run, compared with the communication graph at hand. Namely, in a local final state of process p, the set of previous communication partners of p must coincide with the set of p's direct neighbors in the communication graph.

Let $S^-_{\mathcal{A}}$ abbreviate $\prod_{p\in P}(S_p \uplus \{-\})$ and let F^- be the set of tuples $\overline{s} \in S^-_{\mathcal{A}}$ such that there is $\bar{f} \in F$ such that $\overline{s}$ coincides with $\bar{f}$ in at least two components and, for all other components $p \in P$, it holds both $\overline{s}[p] = -$ and $\bar{f}[p] = \overline{s}^{in}[p]$. Then, $\mathcal{A}'$ is given in detail as follows:

- for any $p \in P$, $S'_p = \Big(S_p \times F^- \times (\prod_{p\in P} 2^P) \times 2^P\Big) \uplus \{\iota_p\}$
  (the first component of a p-local state—apart from ι_p, the p-local initial state—simulates $\mathcal{A}$, while the second component holds a guessed global final state, which, once chosen, cannot be changed anymore; the symbol $-$ shall hereby indicate that the respective process is not expected to move at

all; the third and forth component specify further communication obligations and the set of former communication partners, respectively; hereby, a communication graph $(P(M), \mathit{Arcs})$ (of some MSC M) will be represented by its undirected variant, namely as a tuple $\bar{\vartheta} \in \prod_{p \in P} 2^P$ where, for any $p, q \in P$, $q \in \bar{\vartheta}[p]$ iff both $\{p, q\} \subseteq P(M)$ and $(p, q) \in \mathit{Arcs}$ or $(q, p) \in \mathit{Arcs}$),

– $\mathcal{D}' = \mathcal{D} \times F^- \times \prod_{p \in P} 2^P$
(again, the first component of a message aims at simulating the original MPA, while the remaining components ensure that the guess about a global final state and further communication obligations are respectively transferred to the communication partner),

– for $\theta \in \{!, ?\}$, $(\iota_p, p\theta q, (d, \bar{d}, \bar{\vartheta}_1), (s', \overline{s}', \bar{\vartheta}', \vartheta')) \in \Delta'_p$ if

- $(\overline{s}^{in}[p], p\theta q, d, s') \in \Delta_p$
(of course, the initial local transition must correspond to some initial local transition of $\mathcal{A}$),
- $\bar{\vartheta}'$ represents a connected communication graph such that, for any $r \in P$, $\overline{s}'[r] = -$ implies $\bar{\vartheta}'[r] = \emptyset$
(process p assumes both a global final state $\overline{s}'$ and a communication structure $\bar{\vartheta}'$ and propagates $\overline{s}'$ along $\bar{\vartheta}'$ to ensure an agreement on $\overline{s}'$),
- $(\bar{d}, \bar{\vartheta}_1) = (\overline{s}', \bar{\vartheta}')$,
- $\vartheta' = \{q\}$
(the set of communication partners of p is initialized to $\{q\}$),
- $q \in \bar{\vartheta}[p]$ and, consequently, $p \in \bar{\vartheta}[q]$
(however, the transition can only be taken if this is intended by $\bar{\vartheta}$),

– for $\theta \in \{!, ?\}$, $((s, \overline{s}, \bar{\vartheta}, \vartheta), p\theta q, (d, \bar{d}, \bar{\vartheta}_1), (s', \overline{s}', \bar{\vartheta}', \vartheta')) \in \Delta'_p$ if

- $(s, p\theta q, d, s') \in \Delta_p$
(as above, a transition has to conform to the corresponding local transition relation of $\mathcal{A}$),
- $(\overline{s}, \bar{\vartheta}) = (\bar{d}, \bar{\vartheta}_1) = (\overline{s}', \bar{\vartheta}')$
(as mentioned above, part of the local state cannot be changed anymore; to guarantee an agreement on the global final state $\overline{s}'$, that part is passed/received to/from p's communication partner q, respectively),
- $\vartheta' = \vartheta \cup \{q\}$
(q is added to the set of p's communication partners so far),

- $q \in \bar{\vartheta}[p]$ and, consequently, $p \in \bar{\vartheta}[q]$
  (q is only a valid communication partner of q if this is provided by the communication structure that is represented by $\bar{\vartheta}$),

– $\overline{s}'_0 = (\iota_p)_{p \in P}$
(at the very beginning, no process has made a guess about global final states and the communication structure), and

– $F' = \prod_{p \in P} F'_p$ where, for any $p \in P$, F'_p contains ι_p and, furthermore, any tuple $(s, \overline{s}, \bar{\vartheta}, \vartheta)$ such that both $s = \overline{s}[p]$ and $\vartheta = \bar{\vartheta}[p]$
(unless it did not make any move, process p—more precisely, its current state—has to agree with the global final state and, according to $\bar{\vartheta}$, should have communicated with all the processes it was supposed to do). □

As any MSC over three processes is connected, we get the following:

Corollary 4.4.15 If $|P| \leq 3$, then $\mathcal{L}(\mathrm{MPA}^f_\ell) = \mathcal{L}(\mathrm{MPA}^f)$.

Lemma 4.4.16

$$1\text{-}\mathcal{GA}_{\mathbb{MSC}} \setminus \mathcal{GA}^-_{\mathbb{MSC}} \neq \emptyset$$

Proof Due to Lemma 4.4.13 and Corollary 4.4.15, we cannot expect to find a corresponding language with only three processes. However, it is easy to see that the language $\{G, I\}$ with G and I taken from Figure 3.9 on page 52 is not contained in $\mathcal{GA}^-_{\mathbb{MSC}}$, though it can be recognized by some graph acceptor with 1-spheres and with occurrence constraints. The argument is similar to the one for proving Lemma 2.2.12. □

Theorem 4.4.17 In general, $\mathcal{MSO}[\leq]_{\mathbb{MSC}(P,\Lambda)}$ and $\mathcal{EMSO}_{\mathbb{MSC}(P,\Lambda)}$ are incomparable wrt. inclusion.

Proof Let us provide an MSC language that is $\mathrm{EMSO}_{\mathbb{MSC}}$- but not $\mathrm{MSO}[\leq]_{\mathbb{MSC}}$-definable. This proof direction does not rely on message contents, which are therefore omitted. So consider the MSC language L that contains the MSCs $M(m, n)$, $m, n \in \mathbb{N}$, of the form depicted in Figure 4.12 on the following page. Note that, for any $(E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda) \in L$, $\lhd^*$ is a total order. It is easy to find a finite MPA recognizing L so that L is $\mathrm{EMSO}_{\mathbb{MSC}}$-definable. Now suppose L to be $\mathrm{MSO}[\leq]_{\mathbb{MSC}}$-definable, too. Then, $Lin(L)$ is $\mathrm{MSO}(Act, -)[\leq]_{Lin(\mathbb{MSC})}$-definable, i.e., there is, according to Theorem 2.3.3, a word language $L' \in \mathcal{FA}(Act)$ such that $L' \cap Lin(\mathbb{MSC}) = Lin(L)$. Consider

$$\begin{array}{l}(1!3)^m(1!4)(4?1)(4!1)(1?4)(1!3)^n\\ (1!2)(2?1)(2!3)(3?2)\\ (3?1)^m(3!4)(4?3)(4!3)(3?4)(3?1)^n \in L',\end{array}$$

which corresponds to the MSC from Figure 4.12 (cf. [Kus03]). If m and n are sufficiently large, we can, due to pumping arguments for regular word languages, find a word

$$\begin{array}{l}(1!3)^{(m+k)}(1!4)(4?1)(4!1)(1?4)(1!3)^{n}\\(1!2)(2?1)(2!3)(3?2)\\(3?1)^{m}(3!4)(4?3)(4!3)(3?4)(3?1)^{(n+k)} \in L'\end{array}$$

with $k \geq 1$, which, though it is a valid linearization of some MSC, is not in accord with Figure 4.12 and, therefore, contradicts our premise.

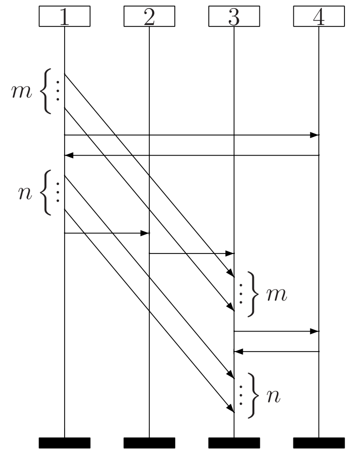


Figure 4.12: An MSC language that is not MSO$[\leq]_{\mathbb{MSC}}$-definable

Conversely, set P to be $\{1, 2\}$ and Λ to be $\{a, b, c\}$. Consider the picture language $L \subseteq \mathbb{P}(\Lambda)$ from the proof of Theorem 2.5.3 and recall that any picture from L can be (uniquely) partitioned into pictures G, C, and H such that the sets of different column labelings of G and H coincide. The unique partition of the picture from Figure 2.8 on page 35 is illustrated in Figure 2.9 on page 36. Such a picture can be encoded as an MSC over (P, Λ) as follows: In the initialization phase, process 1 sends arbitrarily many messages to process 2. Thereupon, any message received is acknowledged immediately until one of the processes stops acknowledging. Those send events that take place during the initialization phase correspond to the first column of the picture to be encoded and are accordingly labeled. Thus, the initialization phase also constitutes the picture's column length, say n in the following. The second column is then given by the first n send events on the second process line, the third column by the second n send events of process 1 and so on. Note that only a send event corresponds to some point in the picture at hand. In this sense, Figure 2.8 gives rise to the MSC from Figure 4.13 on the next page. We omit here a formal definition of such a folding, as, though slightly

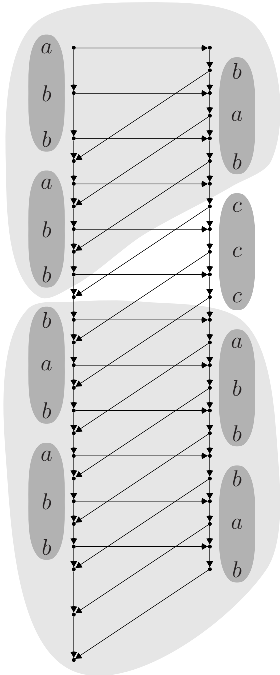


Figure 4.13: Folding a picture

modified, it is given in the following chapter. Now suppose the MSC language L', which consists of all foldings of pictures from L, to be the language of some finite MPA. Then, due to Lemma 4.4.13 and Corollary 4.4.15 (both adapted to extended MSCs), there is a graph acceptor $\mathcal{B}$ over $(Act(P, \Lambda), P_c)$, say with set of control states Q, such that $L_{\mathbb{MSC}(P,\Lambda)}(\mathcal{B}) = L'$. An accepting run of $\mathcal{B}$, as argued in the proof of Theorem 2.5.3, has to transfer all the information it has about the upper part of the MSC (which corresponds to the first partition of the underlying picture) along the middle part of size $2n$ to the lower section. However, as there are $2^{2^n} - 1$ possible distinct nonempty sets of words over $\{a, b\}$ of length n but only $|Q|^{2n}$ possible assignments of states to the middle part, we can, provided n is sufficiently large, find an accepting run of $\mathcal{B}$ on some MSC whose upper and lower part does not fit together in the sense stipulated by L.

It remains to show that L' is $\mathrm{MSO}[\leq]_{\mathbb{MSC}(P,\Lambda)}$-definable. First of all, the set of all foldings of pictures from L is $\mathrm{MSO}[\leq]_{\mathbb{MSC}(P,\Lambda)}$-definable. The corresponding formula basically claims the existence of a chain that starts at the minimal event and alternates between the processes 1 and 2. Moreover, both $\leq_1$ and $\leq_2$, which are not to be confused with the orderings induced by the processes but correspond to walking in the grid from top to bottom and from left to right, respectively, are $\mathrm{MSO}[\leq]_{\mathbb{MSC}(P,\Lambda)}$-definable. For example, $x \leq_2 y$, which stands for proceeding from left to right in the corresponding grid, asks for the existence of a chain starting at x, iterating between 1 and 2, and ending in y. When defining $\leq_1$, the main difficulty is to determine a predicate that marks those events x that correspond to the end of a column. Again, this can be reduced to the existence of a chain starting at x and ending in the greatest event of the folding. Filling in the proof details yields a defining formula for L'. □

Note that we will see in the following chapter (Corollary 5.2.5) that even the classes $\mathcal{MSO}[\leq]_{\mathbb{MSC}(P)}$ and $\mathcal{EMSO}_{\mathbb{MSC}(P)}$ (thus, neglecting any message contents) are incompatible wrt. inclusion.

4.4.6 MPAs vs. EMSO-definable Product Languages

One might expect that, implementing (weak) EMSO-definable product languages, one can do with only one extra message. But appearances are deceiving:

Lemma 4.4.18 We have the following strict inclusions:

(a) $\mathcal{L}(1\text{-MPA}^f_\ell) \subsetneq \mathcal{EP}^0$

(b) $\mathcal{L}(1\text{-MPA}^f) \subsetneq \mathcal{EP}$

Proof Inclusion of (a) follows from Lemma 4.2.1 and Lemma 4.4.3. Inclusion of (b) then proceeds as the proof for Corollary 4.2.2. Let us turn towards strictness. For naturals $m, n \geq 1$, let the MSC $M(m, n)$ be given by its projections $M(m, n) \upharpoonright 1 = (1!2)^m\, ((1?2)(1!2))^n$ and $M(m, n) \upharpoonright 2 = ((2?1)(2!1))^n\, (2?1)^m$. The MSC $M(3, 2)$ is depicted in Figure 4.14 on the facing page. Now consider the EMSO-definable MSC language $L = \{M(n, n) \mid n \geq 1\}$, which is recognized by the locally-accepting finite 2-MPA from Figure 4.1 on page 63 with set of synchronization messages $\{\circ, \bullet\}$. We easily verify that L is a weak product MSC language. However, L is not contained in $\mathcal{L}(1\text{-MPA}^f)$. Because suppose there is an MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in \{1,2\}}, \mathcal{D}, \overline{s}^{in}, F) \in 1\text{-MPA}^f$ with $L(\mathcal{A}) = L$. As $\mathcal{A}$ is finite, there is a natural $n \geq 1$ and an accepting run of $\mathcal{A}$ on $M(n, n)$ such that component $\mathcal{A}_1$, when reading the first n letters 1!2 of $M(n, n) \upharpoonright 1$, goes through a cycle, say of length i (≥ 1), and component $\mathcal{A}_2$, when reading the last n letters 2?1 of $M(n, n) \upharpoonright 2$, goes through another cycle, say of length j (≥ 1). (We just have to choose n large enough.) But then there is also an accepting run of $\mathcal{A}$ on $M(n + (i \cdot j), n) \notin L$, which contradicts the premise. □

Thus, a weak EMSO-definable product language is not necessarily implementable as a finite MPA with one synchronization message. However, this is the case when we restrict to finitely-generated or universally-bounded languages:

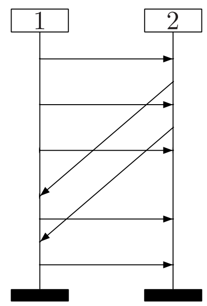


Figure 4.14: $M(3,2)$

Lemma 4.4.19 Let L be a finitely-generated or universally-bounded MSC language.

(a) $L \in \mathcal{EP}^0$ iff $L \in \mathcal{L}(1\text{-MPA}^f_\ell)$.

(b) $L \in \mathcal{EP}$ iff $L \in \mathcal{L}(1\text{-MPA}^f)$.

Proof In first instance, we prove (a). It remains to show the direction from left to right. Let $L \in \mathcal{EP}^0$ and assume L is finitely generated. By means of Theorems 4.1 and 2.3 and Proposition 3.2 from [Mor02], it follows that $L \in \mathcal{L}(1\text{-MPA}^f_\ell)$. Now assume L to be bounded. As L is already EMSO-definable, it is even regular [HMKT00a]. The result then follows from Lemma 4.3.2. Proving (b) proceeds as the proof of Corollary 4.2.2. $\square$

4.5 The Complete Hierarchy

So far, we investigated the expressiveness of MPAs wrt. some language classes proposed in Chapter 3. So let us summarize those results towards a hierarchy of MSC languages.

Theorem 4.5.1 The classes of MSC languages proposed so far draw the picture given by Figure 4.15.

Theorem 4.5.1 follows from results of the preceding sections as well as Lemma 4.5.2 and Lemma 4.5.4.

Lemma 4.5.2 For each $N \geq 1$, $\mathcal{L}((N+1)\text{-}\forall\text{MPA}^f_\ell) \setminus \mathcal{L}(N\text{-MPA}) \neq \emptyset$.

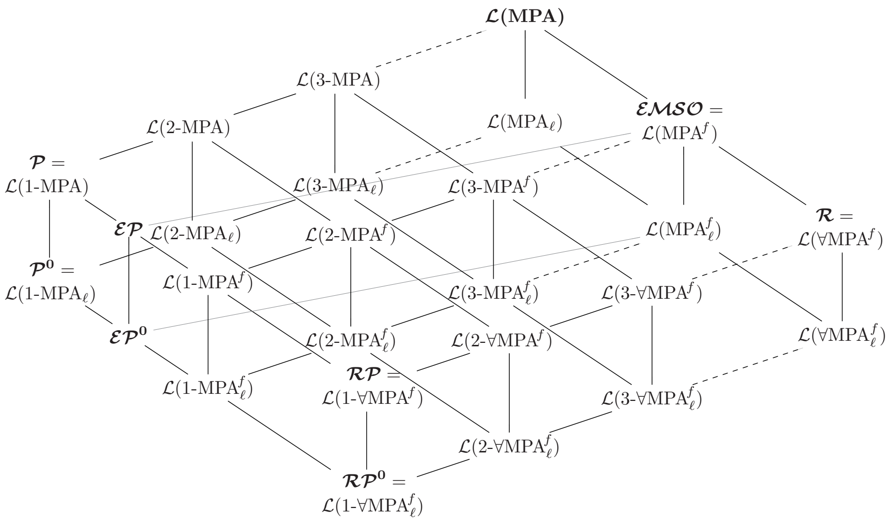


Figure 4.15: A Hierarchy of MSC Languages

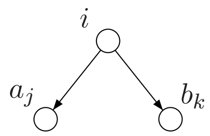


Figure 4.16: Trace $T(i, j, k)$ over $\tilde{\Sigma}_{\widehat{N}}$

Proof We shift the proof into the setting of traces, i.e., we specify a language from $\mathcal{L}((N+1)\text{-}\forall\text{MPA}^f_\ell)$ that turns out to be contained in $\mathcal{L}(N\text{-MPA})$ only if a corresponding trace language is recognized by some product automaton. First of all, however, we show that such a product automaton cannot exist.

Claim 4.5.3 Let $\widehat{N} \geq 1$, let $A_{\widehat{N}+1} = \{a_1, \ldots, a_{\widehat{N}+1}\}$ and $B_{\widehat{N}+1} = \{b_1, \ldots, b_{\widehat{N}+1}\}$ be alphabets, and let the distributed alphabet $\tilde{\Sigma}_{\widehat{N}}$ be given by its components $\Sigma_1 = A_{\widehat{N}+1} \cup \{1, \ldots, \widehat{N}\}$ and $\Sigma_2 = B_{\widehat{N}+1} \cup \{1, \ldots, \widehat{N}\}$. For $i \in \{1, \ldots, \widehat{N}\}$ and $j, k \in \{1, \ldots, \widehat{N}+1\}$, set furthermore the trace $T(i, j, k)$ over $\tilde{\Sigma}_{\widehat{N}}$ to be the one illustrated in Figure 4.16 and the trace language $\mathcal{T}$ to be $\{T(i, j, j) \mid i \in \{1, \ldots, \widehat{N}\} \text{ and } j \in \{1, \ldots, \widehat{N}+1\}\}^*$. There is no product automaton $\mathcal{A}$ over $\tilde{\Sigma}_{\widehat{N}}$ such that $L(\mathcal{A}) = \mathcal{T}$ (even if we allow infinite local state spaces).

Proof of Claim 4.5.3. We prove the claim by contradiction. So suppose there is a product automaton $\mathcal{A} = ((\mathcal{A}_k)_{k=1,2}, \overline{s}^{in}, F)$ over $\tilde{\Sigma}_{\widehat{N}}$, $\mathcal{A}_k = (S_k, \Delta_k)$, with $L(\mathcal{A}) = \mathcal{T}$. For $k = 1, 2$, we define 2^{S_k}-labeled *trees* $t_k = (Dom_k, val_k)$ where Dom_k, the set of *nodes*, is the least set satisfying the following:

- $\varepsilon \in Dom_k$
- if $w \in Dom_k$ and $|w|$ is even, then $\{w1, \ldots, w\widehat{N}\} \subseteq Dom_k$
- if $w \in Dom_k$ and $|w|$ is odd, then $\{w1, \ldots, w(\widehat{N}+1)\} \subseteq Dom_k$

The *valuation function* val_k is given by

- $val_k(\varepsilon) = \{\overline{s}^{in}[k]\}$
- $val_k(wi) = \begin{cases} \{s \in S_k \mid \exists s' \in val_k(w) : \ (s', i, s) \in \Delta_k\} & \text{if } |wi| \text{ is odd} \\ \{s \in S_k \mid \exists s' \in val_k(w) : \ (s', a_i, s) \in \Delta_k\} & \text{if } |wi| \text{ is even} \\ & \text{and } k = 1 \\ \{s \in S_k \mid \exists s' \in val_k(w) : \ (s', b_i, s) \in \Delta_k\} & \text{if } |wi| \text{ is even} \\ & \text{and } k = 2 \end{cases}$

Let $i \geq 1$ be a natural and let furthermore $u = u(1)u(2)\ldots u(2i) \in Dom_1$ and $v = v(1)v(2)\ldots v(2i) \in Dom_2$ be nodes of t_1 and t_2, respectively, such that $val_1(u) \neq \emptyset$ and $val_2(v) \neq \emptyset$. While u stands for a set of computations of $\mathcal{A}_1$, each ending in a state from $val_1(u)$, v represents some computations of $\mathcal{A}_2$. As $\mathcal{A}_1$ and $\mathcal{A}_2$ communicate on the set $\{1, \ldots, \widehat{N}\}$, the pair (u, v) represents a set of runs of $\mathcal{A}$ on $T(u(1), u(2), v(2)) \cdot \ldots \cdot T(u(2i-1), u(2i), v(2i))$ if, for each $j \in \{1, \ldots, i\}$, $u(2j-1) = v(2j-1)$. (Of course, only those pairs (u, v) with $u(2j) = v(2j)$ for each $j \in \{1, \ldots, i\}$ can contain accepting ones.) Let $d \in \mathbb{N}$ such that $(\widehat{N}+1)^d > |F|$ (thus, $d \geq 1$). We can identify at least $(\widehat{N}^2 + \widehat{N})^d$ pairwise distinct nodes w of length $|w| = 2d$ with $(val_1(w) \times val_2(w)) \cap F \neq \emptyset$. This is because the runs (u, v) of $\mathcal{A}$ of length $|u| = |v| = 2d$ accept $(\widehat{N}^2 + \widehat{N})^d$ distinct traces (isomorphism classes of traces). More precisely, there is a set V of nodes of t_1 (which are also nodes of t_2) such that $|V| = (\widehat{N}^2 + \widehat{N})^d$ and, for each $w \in V$, $|w| = 2d$ and $(val_1(w) \times val_2(w)) \cap F \neq \emptyset$. Up to level $2d$, there are $\widehat{N}^d$ possible alternatives to choose successor nodes on an even level. Thus, there is a subset V' of V such that

- $|V'| = \frac{(\widehat{N}^2+\widehat{N})^d}{\widehat{N}^d} = (\widehat{N}+1)^d$ $(> |F|)$ and
- for any $u = u(1)u(2)\ldots u(2d) \in V'$, $v = v(1)v(2)\ldots v(2d) \in V'$, and $j \in \{1, \ldots, d\}$, $u(2j-1) = v(2j-1)$.

For any two nodes $u = u(1)u(2)\ldots u(2d) \in V'$ and $v = v(1)v(2)\ldots v(2d) \in V'$ with $u \neq v$, (u, v) represents, according to the latter item, a set of runs on $T(u(1), u(2), v(2)) \cdot \ldots \cdot T(u(2d-1), u(2d), v(2d))$ with $u(2j) \neq v(2j)$ for at least one $j \in \{1, \ldots, d\}$, which must be all rejecting. Thus, we have $(val_1(u) \times$

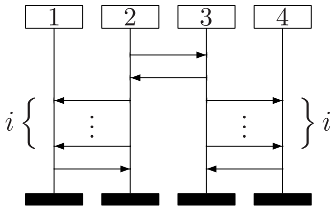


Figure 4.17: MSC $M(i)$

$val_2(v)) \cap F = \emptyset$ for any $u, v \in V'$ with $u \neq v$. But due to $|V'| > |F|$, there are at least two nodes $u, v \in V'$ with $u \neq v$ such that $(val_1(u) \times val_2(v)) \cap F \neq \emptyset$, which contradicts the premise. Thus, a product automaton $\mathcal{A}$ over $\tilde{\Sigma}_{\widehat{N}}$ satisfying $L(\mathcal{A}) = \mathcal{T}$ cannot exist. This concludes the proof of Claim 4.5.3. ■

Let $N \geq 1$ and set $\widehat{N} = N^2$. For $i \in \{1, \ldots, \widehat{N}+1\}$, consider the MSC $M(i)$ as illustrated in Figure 4.17. We claim that the set $L_{\widehat{N}+1} = \{M(i) \mid i \in \{1, \ldots, \widehat{N}+1\}\}^*$ is contained in $\mathcal{L}((N{+}1)\text{-}\forall\mathrm{MPA}_\ell^f) \setminus \mathcal{L}(N\text{-MPA})$. Note first that $L_{\widehat{N}+1}$ is $\forall(\widehat{N}+1)$-bounded. The $\forall$-bounded locally-accepting finite $(N+1)$-MPA recognizing $L_{\widehat{N}+1}$ simply sends from process 2 to process 3 a message with content $n_1 \in \{1, \ldots, N+1\}$ and then receives a message from process 3 with content $n_2 \in \{1, \ldots, N+1\}$. The possible outcomes of (n_1, n_2) now encode the number of messages to be sent both from 2 to 1 and from 3 to 4, respectively. As $(N+1)^2 \geq N^2 + 1$, this is indeed possible using $N+1$ messages. However, restricting to N messages, such an encoding turns out to be no longer achievable. We now show that $L_{\widehat{N}+1}$ in fact cannot be recognized by an N-MPA. Suppose there is an N-MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in \{1,2,3,4\}}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_p = (S_p, \Delta_p)$, with $L(\mathcal{A}) = L_{\widehat{N}+1}$. Without loss of generality, we assume that $\mathcal{D} = \{1, \ldots, N\}$. Let in the following $h : \{1, \ldots, N\}^2 \to \{1, \ldots, \widehat{N}\}$ be a bijective mapping and let, as in the proof of Claim 4.5.3, $A_{\widehat{N}+1} = \{a_1, \ldots, a_{\widehat{N}+1}\}$ and $B_{\widehat{N}+1} = \{b_1, \ldots, b_{\widehat{N}+1}\}$ be alphabets and the distributed alphabet $\tilde{\Sigma}_{\widehat{N}}$ be given by $\Sigma_1 = A_{\widehat{N}+1} \cup \{1, \ldots, \widehat{N}\}$ and $\Sigma_2 = B_{\widehat{N}+1} \cup \{1, \ldots, \widehat{N}\}$. We define $\Longrightarrow_1 \subseteq (S_1 \times S_2) \times \Sigma_1 \times (S_1 \times S_2)$ and $\Longrightarrow_2 \subseteq (S_3 \times S_4) \times \Sigma_2 \times (S_3 \times S_4)$ to be the least sets, respectively, satisfying the following:

- if $(s_2, (2!3, n_1)(2?3, n_2), s_2') \in \Delta_2$ for some $n_1, n_2 \in \{1, \ldots, N\}$ and $s_2, s_2' \in S_2$ (we extend Δ_2 in the obvious manner), then $(s_1, s_2) \overset{h(n_1,n_2)}{\Longrightarrow}_1 (s_1, s_2')$ for each $s_1 \in S_1$

- if $(s_3, (3?2, n_1)(3!2, n_2), s_3') \in \Delta_3$ for some $n_1, n_2 \in \{1, \ldots, N\}$ and $s_3, s_3' \in S_3$, then $(s_3, s_4) \overset{h(n_1,n_2)}{\Longrightarrow}_2 (s_3', s_4)$ for each $s_4 \in S_4$

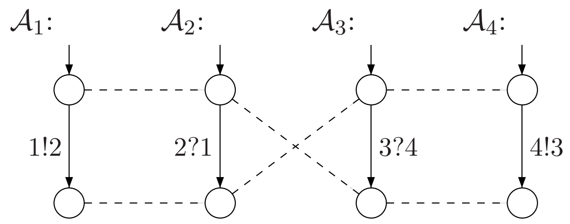


Figure 4.18: Finite 1-MPA recognizing L

– if

$$(s_2, (2!1, i_1) \dots (2!1, i_k)(2?1, i_{k+1}), s_2') \in \Delta_2 \text{ and}$$
$$(s_1, (1?2, i_1) \dots (1?2, i_k)(1!2, i_{k+1}), s_1') \in \Delta_1$$

for some $k \in \{1, \dots, \widehat{N} + 1\}$, $i_1, \dots, i_k, i_{k+1} \in \{1, \dots, N\}$, $s_2, s_2' \in S_2$, and $s_1, s_1' \in S_1$, then $(s_1, s_2) \stackrel{a_k}{\Longrightarrow}_1 (s_1', s_2')$

– if

$$(s_3, (3!4, i_1) \dots (3!4, i_k)(3?4, i_{k+1}), s_3') \in \Delta_3 \text{ and}$$
$$(s_4, (4?3, i_1) \dots (4?3, i_k)(4!3, i_{k+1}), s_4') \in \Delta_4$$

for some $k \in \{1, \dots, \widehat{N} + 1\}$, $i_1, \dots, i_k, i_{k+1} \in \{1, \dots, N\}$, $s_3, s_3' \in S_3$, and $s_4, s_4' \in S_4$, then $(s_3, s_4) \stackrel{b_k}{\Longrightarrow}_2 (s_3', s_4')$

Employing the transition relations $\Longrightarrow_1$ and $\Longrightarrow_2$, we construct a product automaton $\mathcal{A}' = ((\mathcal{A}_i')_{i=1,2}, \overline{s}^{in\prime}, F')$ over $\tilde{\Sigma}_{\widehat{N}}$, $\mathcal{A}_i' = (S_i', \Longrightarrow_i)$ (with possibly infinite S_i'), where

- $S_1' = S_1 \times S_2$ and $S_2' = S_3 \times S_4$,
- $\overline{s}^{in\prime} = ((\overline{s}^{in}[1], \overline{s}^{in}[2]), (\overline{s}^{in}[3], \overline{s}^{in}[4]))$, and
- $F' = \{((s_1, s_2), (s_3, s_4)) \mid (s_1, s_2, s_3, s_4) \in F\}$.

We easily verify that $L(\mathcal{A}) = L_{\widehat{N}+1}$ implies $L(\mathcal{A}') = \mathcal{T}$. But, as Claim 4.5.3 states, such a product automaton $\mathcal{A}'$ does not exist, resulting in a contradiction. $\square$

Lemma 4.5.4 $\mathcal{L}(1\text{-}\forall\text{MPA}^f) \setminus \mathcal{L}(\text{MPA}_\ell) \neq \emptyset$

Proof Let L consist of the MSCs G and I given by Figure 3.9 on page 52. Then, L is contained in $\mathcal{L}(1\text{-}\forall\text{MPA}^f) \setminus \mathcal{L}(\text{MPA}_\ell)$. As processes 1 and 2 do not receive any message from 3 and 4 and vice versa, any locally-accepting MPA accepting both

G and I will accept $G \cdot I$. In contrast, the $\forall$-bounded finite 1-MPA from Figure 4.18 recognizing L has some global knowledge employing global final states. $\square$

We did not pay special attention to the relation between (weak) EMSO-definable product languages and the classes of languages defined by (locally-accepting) finite N-MPAs for $N \geq 2$, which is indicated by the light-gray line in Figure 4.15 on page 102. However, we assume that it is possible to show incomparability respectively witnessed by a language depending on N and similar, even though more complicated, to the one suggested in the proof of Lemma 4.4.18.

Chapter 5

Beyond Implementability

In this chapter, we turn our attention to the relation between MSO logic over MSCs and its existential fragment (and therefore, implicitly, finite MPAs, which refer to the notion of implementability). We also compare those logics to the classes of rational and recognizable MSC languages. We show that MSO logic is strictly more expressive than EMSO. Together with the results of the previous chapter, this will be used to prove that finite MPAs cannot be complemented in general, solving an open problem raised by Kuske [Kus01, Kus03], and that they cannot be determinized. Those results rely on an encoding of grids into MSCs, which allows to apply results from the framework of grids and graphs in general to MSCs. Altogether, we highlight the application limitations of MPAs so that future work might aim at finding large classes of (finite) MPAs that still have promising algorithmic properties.

5.1 EMSO vs. MSO in the Bounded Setting

Let us first recall the corresponding problem in the bounded setting where we restrict the interpretation of formulas to $\forall B$-bounded MSCs.

Theorem 5.1.1 For any $B \geq 1$,

$$\mathcal{EMSO}_{\mathbb{MSC}_{\forall B}} = \mathcal{MSO}_{\mathbb{MSC}_{\forall B}} = \mathcal{EMSO}[\leq]_{\mathbb{MSC}_{\forall B}} = \mathcal{MSO}[\leq]_{\mathbb{MSC}_{\forall B}}.$$

Proof First note that the language of a finite MPA restricted to the set of $\forall B$-bounded MSCs is regular. More precisely, for any finite MPA $\mathcal{A}$ and any $B \geq 1$, $L(\mathcal{A}) \cap \mathbb{MSC}_{\forall B}$ is a regular MSC language. With Theorem 4.4.1 and results from [HMKT00b] and [Kus03], this implies $\mathcal{EMSO}_{\mathbb{MSC}_{\forall B}} = \mathcal{EMSO}[\leq]_{\mathbb{MSC}_{\forall B}} = \mathcal{MSO}[\leq]_{\mathbb{MSC}_{\forall B}}$.

It remains to show that $\mathcal{EMSO}_{\mathbb{MSC}_{\forall B}} \supseteq \mathcal{MSO}_{\mathbb{MSC}_{\forall B}}$. More generally, we show that, for each $\varphi(Y_1, \ldots, Y_i) \in$ MSO, $i \geq 1$, there is a $\forall$-bounded finite MPA $\mathcal{A}$ (adapted to structures from $\mathbb{MSC}^{\{0,1\}^i}$) such that $L(\mathcal{A}) = L_{\mathbb{MSC}_{\forall B}}(\varphi)$. So let $\varphi(Y_1, \ldots, Y_i) \in$ MSO, which, according to Lemma 2.2.4, can be assumed to be of the form

$$\exists \overline{X_k} \forall \overline{X_{k-1}} \ldots \exists/\forall \overline{X_1} \psi(Y_1, \ldots, Y_i, \overline{X_k}, \ldots, \overline{X_1})$$

or, equivalently,

$$\exists \overline{X_k} \neg \exists \overline{X_{k-1}} \ldots \neg \exists \overline{X_1} \psi'(Y_1, \ldots, Y_i, \overline{X_k}, \ldots, \overline{X_1})$$

for some $k \geq 1$. We proceed by induction on k. For $k = 1$, φ is an EMSO-formula, which has an equivalent MPA^f-counterpart $\mathcal{A}$ (tailored to extended MSCs), i.e., $L(\mathcal{A}) = L_{\mathbb{MSC}}(\varphi)$. Using $\Longrightarrow_{\mathcal{A}}$, we gain some finite automaton over $Act \times \{0,1\}^i$ recognizing $Lin(L(\mathcal{A}) \cap (\mathbb{MSC}_{\forall B})^{\{0,1\}^i})$, which is a witness for the fact that $L(\mathcal{A}) \cap (\mathbb{MSC}_{\forall B})^{\{0,1\}^i}$ is a regular MSC language. According to [MKS00], there is $\mathcal{A}' \in$ det-$\forall$MPA^f with $L(\mathcal{A}') = L(\mathcal{A}) \cap (\mathbb{MSC}_{\forall B})^{\{0,1\}^i}$. (Though we did not explicitly define what determinism means for extended MPAs, it is obvious how to adjust the definition accordingly so that, then, the abovementioned result by Mukund et al. also holds in the extended setting.) Induction now alternately involves complementation and projection steps. A complementation step first requires to construct from $\mathcal{A}'$ the finite MPA $\overline{\mathcal{A}'}$ with $L(\overline{\mathcal{A}'}) = \mathbb{MSC}^{\{0,1\}^i} \setminus L(\mathcal{A}')$, which, though taking into account that we deal with extended MSCs, can be found along the usual lines, i.e., first providing a complete deterministic MPA, whose set of global final states is then complemented. Projection is even easier, as each communication action just needs to be projected onto the remaining components. □

Thus, our work extends the results by Henriksen et al. [HMKT00b] and Kuske [Kus03], which will be slightly generalized in the following. Moreover, as sets of $\forall B$-bounded MSCs can be seen as sets of Mazurkiewicz traces [Kus03] and, in the setting of Mazurkiewicz traces, MSO logic is expressively equivalent to asynchronous automata (cf. Theorem 2.4.10), Theorem 4.4.1 can be understood as an extension of Zielonka's Theorem.

Proposition 5.1.2 ([Kus03]) For any $B \geq 1$, the following hold:

(a) $\mathbb{MSC}_{\forall B} \in \mathcal{EMSO}_{\mathbb{MSC}}$

(b) $\mathbb{MSC}_{\forall B} \in \mathcal{EMSO}[\leq]_{\mathbb{MSC}}$

By Proposition 5.1.2, Theorem 5.1.1 can be sharpened as follows:

Theorem 5.1.3 For any $\forall$-bounded MSC language L, the following statements are equivalent:

1. $L \in \mathcal{EMSO}_{\mathbb{MSC}}$
2. $L \in \mathcal{MSO}_{\mathbb{MSC}}$
3. $L \in \mathcal{EMSO}[\leq]_{\mathbb{MSC}}$
4. $L \in \mathcal{MSO}[\leq]_{\mathbb{MSC}}$
5. $L \in \mathcal{MPA}$

Recently, it was even shown that, if we restrict to $\exists$-bounded MSC languages, any $\mathrm{MSO}_{\mathbb{MSC}}$-definable set is implementable.

Theorem 5.1.4 ([Gen04, GKM04]) Theorem 5.1.3 holds for $\exists$-bounded MSC languages verbatim.

The proof by Genest et al. makes use of ideas from [Kus03]: existentially-bounded MSC languages are also seen as trace languages, which allows to apply certain *asynchronous mappings* [DR95].

5.2 EMSO vs. MSO in the Unbounded Setting

In this section, we show that, in contrast to the bounded case (no matter if globally or existentially, as we have seen), quantifier alternation forms a hierarchy, i.e., MSO over MSCs is strictly more expressive than the most expressive model of a finite MPA.

Matz and Thomas proved infinity of the monadic quantifier-alternation hierarchy over grids [MT97, Tho97a] (cf. Theorem 2.5.4). We show how grids can be encoded into MSCs and then rewrite their result in terms of MSCs adapting their proof to our setting.

Theorem 5.2.1 The monadic quantifier-alternation hierarchy over $\mathbb{MSC}$ is infinite.

Proof A grid $G(n, m)$ can be folded to an MSC $M(n, m)$ as exemplarily shown for $G(3, 5)$ in Figure 5.1 on the following page. A similar encoding was used in [Tho96] to transfer results on grids to the setting of acyclic graphs with bounded antichains. By the type of an event, we recognize which events really correspond to a node of the grid, namely those that are labeled with a send action performed by process 1 or 2. Formally, $M(n,m)$ is given by its projections as follows:

$$M(n,m)\upharpoonright 1 \;=\; \begin{cases} (1!2)^n\,[(1?2)(1!2)]^{n((m-1)/2)} & \text{if } m \text{ is odd} \\ (1!2)^n\,[(1?2)(1!2)]^{n((m/2)-1)}\,(1?2)^n & \text{if } m \text{ is even} \end{cases}$$

$$M(n,m)\upharpoonright 2 \;=\; \begin{cases} [(2?1)(2!1)]^{n((m-1)/2)}\,(2?1)^n & \text{if } m \text{ is odd} \\ [(2?1)(2!1)]^{n(m/2)} & \text{if } m \text{ is even} \end{cases}$$

A grid language $\mathcal{G}$ defines the MSC language $L(\mathcal{G}) := \{M(n,m) \mid G(n,m) \in \mathcal{G}\}$. For a function $f : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$, we furthermore write $L(f)$ as a shorthand for the MSC language $L(\mathcal{G}(f))$. We now closely follow [Tho97a], which resumes the result of [MT97]. So let, for $k \in \mathbb{N}$, the functions $s_k, f_k : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ be inductively defined via $s_0(n) = n$, $s_{k+1}(n) = 2^{s_k(n)}$, $f_0(n) = n$, and $f_{k+1}(n) = f_k(n) \cdot 2^{f_k(n)}$.

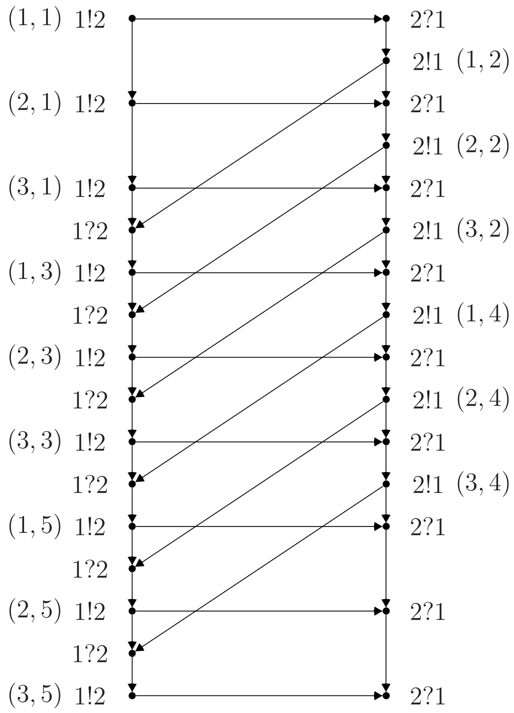


Figure 5.1: Folding the $(3,5)$-grid

Claim 5.2.2 For each $k \in \mathbb{N}$, the MSC language $L(f_k)$ is $(\Sigma_{2k+3})_{\mathbb{MSC}}$-definable.

Proof of Claim 5.2.2. We will show that, for any $k \geq 1$, if a grid language $\mathcal{G}$ is $(\Sigma_k)_{\mathbb{GR}}$-definable (over grids), then $L(\mathcal{G})$ is $(\Sigma_k)_{\mathbb{MSC}}$-definable (over MSCs).

The claim then follows from the fact that any grid language $\mathcal{G}(f_k)$ is $(\Sigma_{2k+3})_{\mathbb{GR}}$-definable [Tho97a]. So let $k \in \mathbb{N}_{\geq 1}$. Figure 5.3 on page 113 shows the MPA $\mathcal{A}_{\mathcal{GF}}$, which recognizes the set of all possible grid foldings. For clarity, ε-transitions are employed, which can be easily eliminated without affecting the recognized language. As usual, a global final state is depicted by a dashed line. Moreover, its labeling indicates, which grid foldings it accepts, while n and m range over $\mathbb{N}_{\geq 1}$ and $\mathbb{N}$, respectively. Alternatively, a corresponding EMSO-sentence requires the existence of a chain iterating between processes 1 and 2. So let, according to Lemma 4.4.3, $\varphi_{\mathcal{GF}} = \exists \overline{X} \psi_{\mathcal{GF}}(\overline{X})$ be an EMSO-sentence (over MSCs) with first-order kernel $\psi_{\mathcal{GF}}(\overline{X})$ that defines the language of $\mathcal{A}_{\mathcal{GF}}$. Moreover, let $\varphi = \exists \overline{Y_1} \forall \overline{Y_2} \ldots \exists/\forall \overline{Y_k} \varphi'(\overline{Y_1}, \ldots, \overline{Y_k})$ be a Σ_k-sentence (over grids) where $\varphi'(\overline{Y_1}, \ldots, \overline{Y_k})$ contains no set quantifiers. Without loss of generality, $\varphi_{\mathcal{GF}}$ and φ employ distinct sets of variables, which, moreover, are supposed to be different from some variable Z. We now determine the Σ_k-sentence Ψ_φ over MSCs with $L_{\mathbb{MSC}}(\Psi_\varphi) = L(L_{\mathbb{GR}}(\varphi))$, i.e., the foldings of $L_{\mathbb{GR}}(\varphi)$ form exactly the MSC language defined by Ψ_φ. Namely, Ψ_φ is given by

$$\exists Z \exists \overline{X} \exists \overline{Y_1} \forall \overline{Y_2} \ldots \exists/\forall \overline{Y_k} (\psi_{bottom}(Z) \wedge \psi_{\mathcal{GF}}(\overline{X}) \wedge \|\varphi'(\overline{Y_1}, \ldots, \overline{Y_k})\|_Z).$$

Hereby, the first-order formula $\psi_{bottom}(Z)$ with free variable Z makes sure that Z is reserved to those send events that correspond to the end of a column (for simplicity, Z may contain some receive events, too), which are highlighted in Figure 5.2 on the following page for $M(3,5)$. This can be easily formalized starting with the requirement that Z contains the maximal send event on the first process line that is not preceded by some receive event. Furthermore, $\|\varphi'(\overline{Y_1}, \ldots, \overline{Y_k})\|_Z$ is inductively derived from $\varphi'(\overline{Y_1}, \ldots, \overline{Y_k})$ as follows:

- $\|x = y\|_Z = (x = y)$

- $\|S_1(x,y)\|_Z =$
$$\begin{array}{ll} & \neg(x \in Z) \\ \wedge & \bigvee_{\sigma \in \{1!2, 2!1\}} (\lambda(x) = \sigma \wedge \lambda(y) = \sigma) \\ \wedge & \qquad x \lhd_1 y \\ & \vee \quad \exists z(\lambda(z) = 1?2 \wedge x \lhd_1 z \wedge z \lhd_1 y) \\ & \vee \quad \exists z(\lambda(z) = 2?1 \wedge x \lhd_2 z \wedge z \lhd_2 y) \end{array}$$

- $\|S_2(x,y)\|_Z =$
$$\begin{array}{ll} & \lambda(x) = 1!2 \wedge \lambda(y) = 2!1 \wedge \exists z(x \lhd_{\mathrm{c}} z \wedge z \lhd_2 y) \\ \vee & \lambda(x) = 2!1 \wedge \lambda(y) = 1!2 \wedge \exists z(x \lhd_{\mathrm{c}} z \wedge z \lhd_1 y) \end{array}$$

- $\|x \in X\|_Z = x \in X$

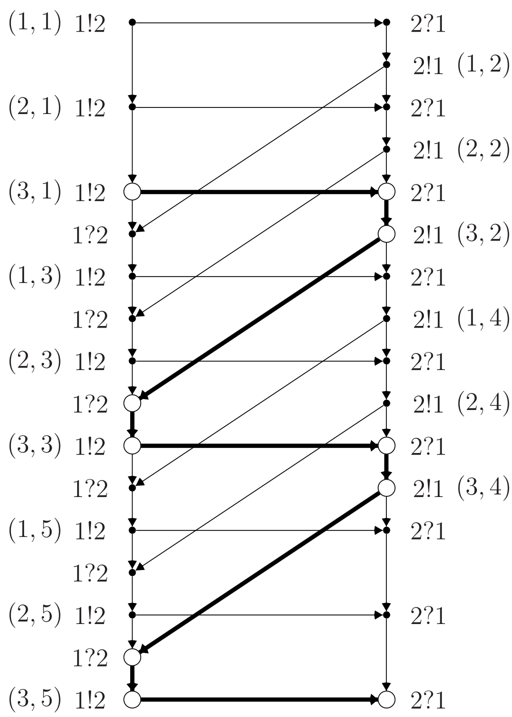


Figure 5.2: Events of a grid folding that correspond to the end of a column

- $\|\neg\varphi\|_Z = \neg\|\varphi\|_Z$
- $\|\varphi \vee \psi\|_Z = \|\varphi\|_Z \vee \|\psi\|_Z$
- $\|\varphi \wedge \psi\|_Z = \|\varphi\|_Z \wedge \|\psi\|_Z$
- $\|\varphi \rightarrow \psi\|_Z = \|\varphi\|_Z \rightarrow \|\psi\|_Z$
- $\|\varphi \leftrightarrow \psi\|_Z = \|\varphi\|_Z \leftrightarrow \|\psi\|_Z$
- $\|\exists x\varphi\|_Z = \exists x((\bigvee_{\sigma\in\{1!2,2!1\}} \lambda(x) = \sigma) \wedge \|\varphi\|_Z)$
- $\|\forall x\varphi\|_Z = \forall x((\bigvee_{\sigma\in\{1!2,2!1\}} \lambda(x) = \sigma) \rightarrow \|\varphi\|_Z)$

Similarly to the proof of Lemma 4.4.3, the above inductive derivation makes sure that only elements that correspond to grid nodes are assigned to $\overline{Y_1}, \ldots, \overline{Y_k}$. ■

Claim 5.2.3 Let $f : \mathbb{N}_{\geq 1} \rightarrow \mathbb{N}_{\geq 1}$ be a function. If $L(f)$ is $(\Sigma_k)_{\mathtt{MSC}}$-definable for some $k \geq 1$, then $f(n)$ is in $s_k(\mathcal{O}(n))$.

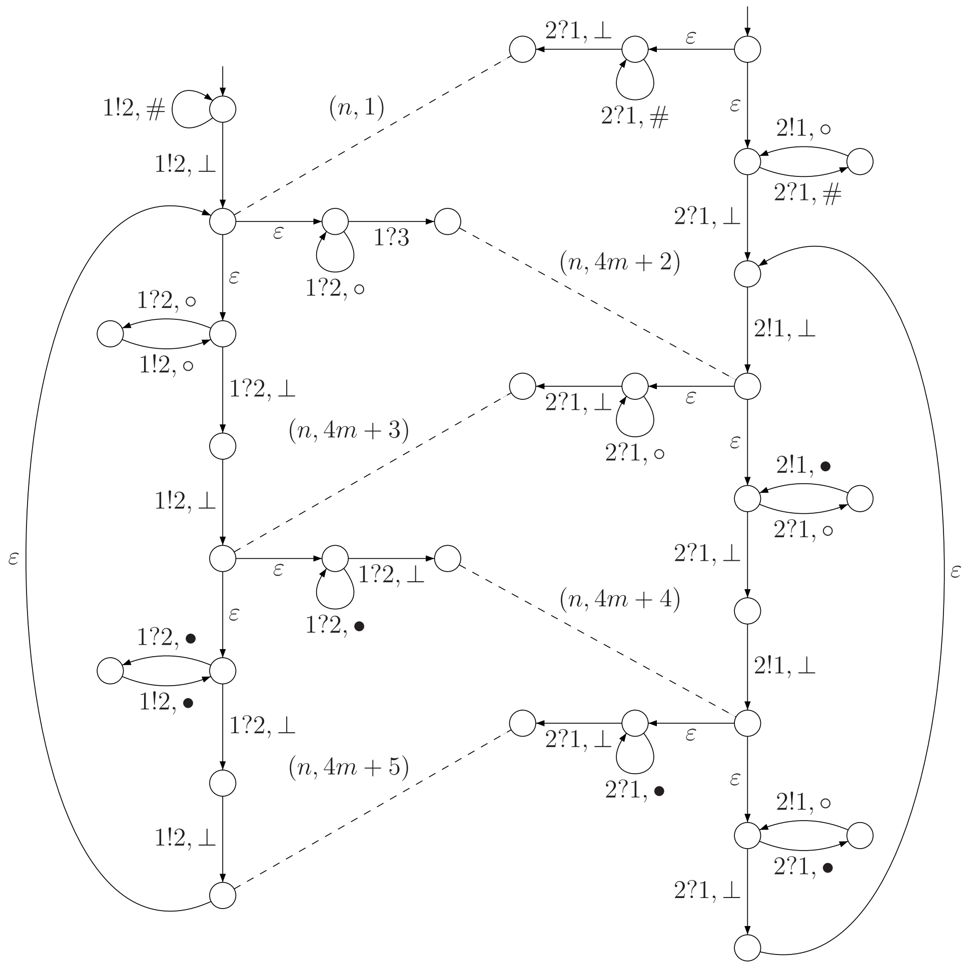


Figure 5.3: A message-passing automaton recognizing $\mathcal{GF}$

Proof of Claim 5.2.3. Let $k \geq 1$ and let in the following the events of an MSC $(E, \{\lhd_p\}_{p\in P}, \lhd_c, \lambda)$ be labeled with elements from $Act \times \{0,1\}^i$ for some $i \in \mathbb{N}_{\geq 1}$, i.e., $\lambda : E \to Act \times \{0,1\}^i$. But note that the type of an event still depends on the type of its communication action only. Let furthermore $\varphi(Y_1, \ldots, Y_i)$ be a Σ_k-formula defining a set of MSCs over the new label alphabet that are foldings of grids. For a fixed column length $n \geq 1$, we will build a finite (word) automaton $\mathcal{A}_n$ over $(Act \times \{0,1\}^i)^n$ with $s_{k-1}(c^n)$ states (for some constant c) that reads grid-folding MSCs column by column and is equivalent to $\varphi(Y_1, \ldots, Y_i)$ wrt. grid foldings with column length n. Column here means a sequence of communication actions, each provided with an additional label, that represents a column in the corresponding grid. For example, running on the MSC $M(3,5)$ as shown in Figure 5.1 on page 110, $\mathcal{A}_3$ first reads the *letter* $(1!2)^3$ (recall that each action is still provided with an extra labeling, which we omit here for the sake of clarity), then continues reading $((2?1)(2!1))^3$ and so on. Then, the shortest word accepted by $\mathcal{A}_n$ has length $\leq s_{k-1}(c^n)$ so that, if $\varphi(Y_1, \ldots, Y_i)$ defines an MSC language $L(f)$ for some f, we have $f(n) \in s_k(\mathcal{O}(n))$. Let us now turn to the construction of $\mathcal{A}_n$. The formula $\varphi(Y_1, \ldots, Y_i)$ is of the form

$$\exists \overline{X_k} \forall \overline{X_{k-1}} \ldots \exists/\forall \overline{X_1} \psi(Y_1, \ldots, Y_i, \overline{X_k}, \ldots, \overline{X_1})$$

or, equivalently,

$$\exists \overline{X_k} \neg \exists \overline{X_{k-1}} \ldots \neg \exists \overline{X_1} \psi'(Y_1, \ldots, Y_i, \overline{X_k}, \ldots, \overline{X_1}).$$

We proceed by induction on k. For $k = 1$, $\varphi(Y_1, \ldots, Y_i)$ is an EMSO-formula. According to Theorem 2.2.13, its MSC language (consisting of MSCs with extended labelings) coincides with the MSC language of some graph acceptor. The transformation from graph acceptors to MPAs from the proof of Theorem 4.4.1 can be easily adapted to handle the extended labeling. Thus, $\varphi(Y_1, \ldots, Y_i)$ defines a language that is recognized by some finite MPA $\mathcal{A} = ((\mathcal{A}_p)_{p\in P}, \mathcal{D}, \overline{s}^{in}, F)$. The automaton $\mathcal{A}_n$ can now be obtained from $\mathcal{A}$ using a part of its global transition relation $\Longrightarrow_{\mathcal{A}} \subseteq \mathit{Conf}_{\mathcal{A}} \times (Act \times \{0,1\}^i) \times \mathcal{D} \times \mathit{Conf}_{\mathcal{A}}$. Note that we have to consider only a bounded number of channel contents, as the set of grid foldings with column length n forms a $\forall n$-bounded MSC language. For some constant c, we have $(|S_{\mathcal{A}}| \cdot (|\mathcal{D}| + 1))^{|Ch| \cdot n} \leq c^n$. Thus, $c^n = s_0(c^n)$ is an upper bound for the number of states of $\mathcal{A}_n$, which only depends on the automaton $\mathcal{A}$ and, thus, on $\varphi(Y_1, \ldots, Y_i)$. The induction steps respectively involve both a complementation step (for negation) and a projection step (concerning existential quantification). While the former increases the number of states exponentially, the latter leaves it constant so that, altogether, the required number of states is obtained. This concludes the proof of Claim 5.2.3. ■

As $f_{k+1}(n)$ is not in $s_k(\mathcal{O}(n))$, it follows from Claims 5.2.2 and 5.2.3 that the hierarchy of classes of $(\Sigma_k)_{\mathbb{MSC}}$-definable MSC languages is infinite. $\square$

Corollary 5.2.4

$$\mathcal{MPA} \;=\; \mathcal{EMSO}_{\mathbb{MSC}} \;\subsetneqq\; \mathcal{MSO}_{\mathbb{MSC}}$$

As, for any $f : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ and $(E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda) \in L(f)$, $\lhd = \lessdot$, which is first-order definable in terms of $\leq$, we obtain the following (cf. Theorem 4.4.17):

Corollary 5.2.5 $\mathcal{MSO}[\leq]_{\mathbb{MSC}}$ and $\mathcal{EMSO}_{\mathbb{MSC}}$ are incomparable wrt. inclusion.

As $\mathcal{MPA} = \mathcal{EMSO}_{\mathbb{MSC}}$, it follows that the complement $\mathbb{MSC} \setminus L$ of an MSC language $L \in \mathcal{MPA}$, is not necessarily contained in $\mathcal{MPA}$, too. Thus, we get the answer to an open question, which has been raised by Kuske [Kus01, Kus03].

Theorem 5.2.6 $\mathcal{MPA}$ is not closed under complementation.

5.3 Determinism vs. Nondeterminism

Real-life distributed systems are usually deterministic. Besides the absence of deadlocks, determinism is therefore one of the crucial properties an implementation of a distributed protocol should have. Previous results immediately affect the question of whether deterministic MPAs suffice to achieve the full expressive power of general MPAs. It is well-known that, in the framework of words and traces, any finite automaton and, respectively, any asynchronous automaton admits an equivalent deterministic counterpart. However, things are more complicated regarding MSCs. Let us first have a look at the bounded setting.

Theorem 5.3.1 ([MKS00, Kus03])

$$\mathcal{L}(\text{det-}\forall\text{MPA}^f) \;=\; \mathcal{L}(\forall\text{MPA}^f)$$

The algorithm by Mukund et al. to construct from a nondeterministic MPA a deterministic counterpart is based on a technique called *time stamping*, while Kuske's construction relies on *asynchronous mappings* for traces.

Unfortunately, the preceding result cannot be transferred to the unbounded setting.

Theorem 5.3.2

$$\mathcal{L}(\text{det-MPA}^f) \;\subsetneqq\; \mathcal{L}(\text{MPA}^f)$$

Proof According to the algorithm from page 63, we can assume a deterministic finite MPA $\mathcal{A} = ((\mathcal{A}_p)_{p\in P}, \mathcal{D}, \overline{s}^{in}, F)$ to be complete in the sense that, for any MSC M, it allows exactly one run on M. If we set $\overline{\mathcal{A}}$ to be the deterministic finite MPA $((\mathcal{A}_p)_{p\in P}, \mathcal{D}, \overline{s}^{in}, S_{\mathcal{A}} \setminus F)$, it holds $L(\overline{\mathcal{A}}) = \mathbb{MSC} \setminus L(\mathcal{A})$. Thus, $\mathcal{L}$(det-MPA^f) is closed under complementation. However, as Theorem 5.2.6 states, $\mathcal{L}$(MPA^f) is not closed under complementation, which implies the theorem. $\square$

Theorems 5.2.6 and 5.3.2 show that both EMSO logic and finite MPAs in their unrestricted form are unlikely to have some nice algorithmic properties that would attract practical interest.

5.4 MPAs vs. Recognizability

In the field of MSCs, recognizability, which was first studied by Morin in [Mor02], turned out to be closely related to implementability.

Proposition 5.4.1 ([Mor02]) For any finitely-generated MSC language L, we have $L \in \mathcal{REC}_{\mathbb{MSC}}$ iff $L \in \mathcal{MSO}[\leq]_{\mathbb{MSC}}$.

Recall that, due to [Gen04, GKM04], every finitely-generated MSC language that is also MSO$[\leq]_{\mathbb{MSC}}$-definable admits an implementation in terms of a finite MPA.

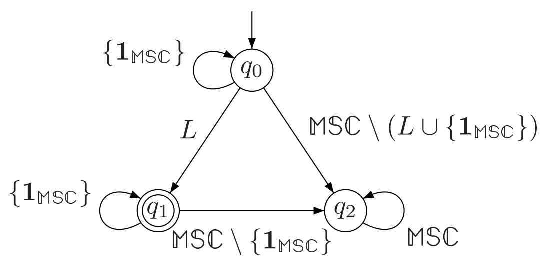


Figure 5.4: An $\mathbb{MSC}$-automaton

Let us now focus on implementability and investigate the expressive power of finite MPAs relative to the class of recognizable MSC languages. In fact, any finite implementation describes a recognizable MSC language, while, however, there are more recognizable languages than implementable ones.

Theorem 5.4.2

$$\mathcal{MPA} \subsetneq \mathcal{REC}_{\mathbb{MSC}}$$

Proof It is easy to provide a recognizable language that is not implementable. In fact, any set L of prime MSCs is recognizable by the $\mathbb{MSC}$-automaton depicted in Figure 5.4 on the facing page. Conversely, suppose $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, \mathcal{D}, \overline{s}^{in}, F)$ to be a finite MPA. For a global state $\overline{s} \in S_{\mathcal{A}}$ of $\mathcal{A}$ and an MSC M, we denote by $\widehat{\delta}(\overline{s}, M)$ the set of global states $\overline{s}' \in S_{\mathcal{A}}$ such that, in the canonical manner, $\mathcal{A}$ admits some run on M that starts in $\overline{s}$ and ends in $\overline{s}'$. Then, $(2^{S_{\mathcal{A}}}, \delta, \{\overline{s}^{in}\}, \{S \subseteq S_{\mathcal{A}} \mid S \cap F \neq \emptyset\})$, where, for any $S \subseteq S_{\mathcal{A}}$ and any MSC M, $\delta(S, M) = \bigcup_{\overline{s} \in S} \widehat{\delta}(\overline{s}, M)$, is an $\mathbb{MSC}$-automaton whose language is $L(\mathcal{A})$. □

Concluding, one might argue that, actually, recognizability makes sense for finitely-generated MSC languages, where it reduces to implementability, only. This view is supported by Proposition 2.1.3, according to which $\mathbb{MSC}$ is not rational. However, the latter property is shared with the class $\mathcal{R}_{\mathbb{MSC}}$ of regular MSC languages, which does not contain $\mathbb{MSC}$ either.

5.5 MPAs vs. Rational MSC Languages

A major goal regarding high-level constructs has been to identify specifications that allow implementations in terms of MPAs, preferably automatically and efficiently. This is difficult in general, as an implementation is controlled locally rather than globally, while high-level descriptions such as HcMSCs allow to declare global states that control multiple processes at the same time. Moreover, it has been studied how to regain from an implementation a specification and whether a formalism is complete in the sense that any implementable language can be specified at all. Those efforts are important with regard to the analysis of a system, as they help to gain new insights in how an implementation is structured. The former question of deriving an implementation from a given specification is raised in [GMSZ02, HK02], for example, while the inverse problem was tackled in [MP01, GMP01]. However, we first show, that completeness is not achievable if we consider classes of HcMSCs.

Proposition 5.5.1 There is an implementable MSC language that cannot be described by some HcMSC.

Proof An implementable MSC language that is not the MSC language of some HcMSC is depicted in Figure 5.5 on the next page where the A_i, B_i, C_i are supposed to be naturals indicating how often a corresponding message is sent. Let us denote this language by L and suppose there is an HcMSC $\mathcal{H}$ whose basic MSC language is L. As A_1 and A_2 can be arbitrarily large, there must occur respective iterations in $\mathcal{H}$, which allow for sending arbitrarily many messages from 2 to 4.

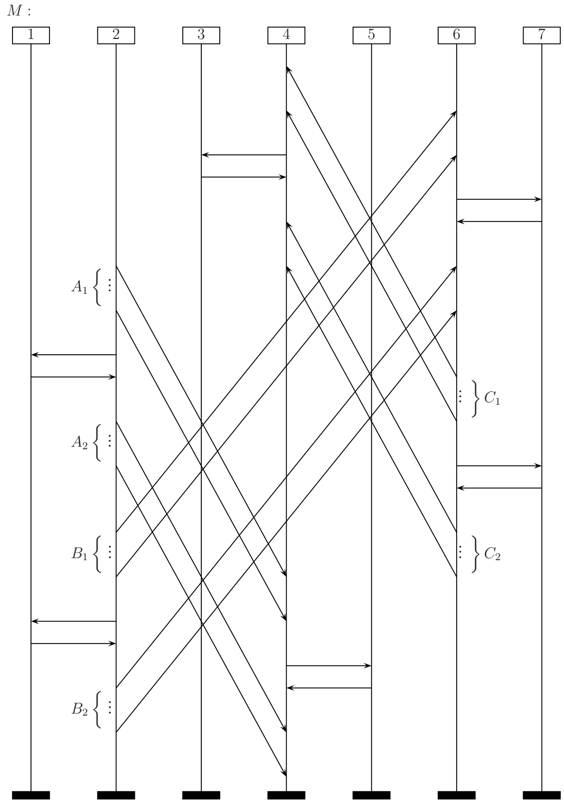


Figure 5.5: An MSC language that is implementable but not the MSC language of some HcMSC

Now suppose that this is only done by partial MSCs that consist of one single send event. Consequently, there must be an iteration of respective single receive events. Due to pumping arguments, those unmatched singletons can be combined towards an MSC where messages sent in the A_1-phase are only received in the A_2-phase, which is not desired. So, when building an MSC, $\mathcal{H}$ must employ at least one partial MSC M_A that contains a complete message transfer from 2 to 4. By the same argument, there have to appear prime MSCs M_B and M_C in $\mathcal{H}$ that represent a *complete* message from 2 to 6 (in the B_1- or B_2-phase) and a message from 6 to 4 (in the C_1- or C_2-phase), respectively. But, obviously, the three MSCs M_A, M_B, and M_C we identified cannot contribute to an MSC as depicted in Figure 5.5, which is a contradiction. $\square$

Generalizing a result by Genest et al., who showed that any safe gc-HcMSC can be defined in terms of $\mathrm{MSO}_{\mathbb{MSC}}$ [Gen04], we show that any gc-HcMSC describes an $\mathrm{MSO}_{\mathbb{MSC}}$-definable MSC language.

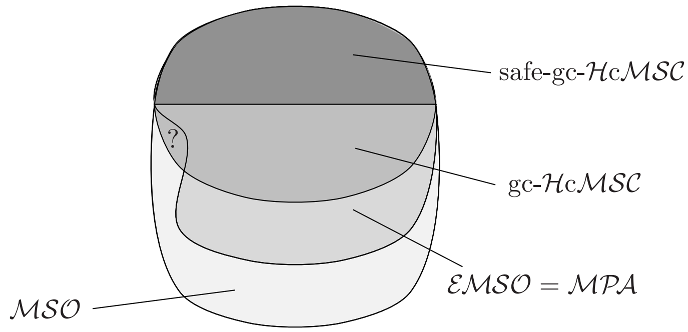


Figure 5.6: An overview

Theorem 5.5.2

$$\text{gc-}\mathcal{H}\text{c}\mathcal{MSC} \subseteq \mathcal{MSO}_{\mathbb{MSC}}$$

Proof Let $\mathcal{H}$ be a gc-HcMSC. Without loss of generality, we can assume that any nonempty partial MSC that occurs in $\mathcal{H}$ is prime and that there is at least one prime partial MSC in $\mathcal{H}$. Now set $\Gamma = \{a_1, \ldots, a_m\}$ to be the (finite) set of those prime partial MSCs, which gives rise to the trace monoid $\mathbb{TR}(\tilde{\Sigma}_\Gamma)$. Let α be a copy of $\mathcal{H}$, which will then be seen as a rational expression of $\mathbb{TR}(\tilde{\Sigma}_\Gamma)$. According to Section 2.4, we can assume the existence of an MSO$(\Gamma, -)$-sentence φ that defines $L(\alpha)$ relative to $\mathbb{TR}(\tilde{\Sigma}_\Gamma)$, i.e., $L(\alpha) = L_{\mathbb{TR}(\tilde{\Sigma}_\Gamma)}(\varphi)$. We now construct a formula $\Phi \in \text{MSO}(Act, P_c)$ such that $\mathcal{L}(\mathcal{H}) = L_{\mathbb{MSC}}(\Phi)$ as follows:

We extend Γ towards $\Gamma' = \{a_1, \ldots, a_m, a'_1, \ldots, a'_m\}$, which contains for each member a_i of Γ a distinct copy a'_i. Moreover, we define a new distributed alphabet $\tilde{\Sigma}'_\Gamma$ to be $(\Sigma'_p)_{p \in P}$ where, for any $p \in P$, $\Sigma'_p = \{a \in \Gamma' \mid p \in P(a)\}$. It is not hard to see that the trace language that we obtain from $L(\alpha)$ if we replace in any trace every second a_i with its copy a'_i is $\text{MSO}(\Gamma', -)_{\mathbb{TR}(\tilde{\Sigma}'_\Gamma)}$-definable, say by a sentence α'. In particular, for any *trace* $(E, \lhd, \lambda) \in L(\alpha')$ and $e, e' \in E$, $e \lhd e'$ implies $\lambda(e) \neq \lambda(e')$. Though a'_i is a distinct copy of a_i, it will stand for the same prime partial MSC as a_i. This will prove useful when we now transform α' into the formula $\Phi \in \text{MSO}(Act, P_c)$ that defines $\mathcal{L}(\mathcal{H})$ relative to $\mathbb{MSC}$. It is basically of the form

$$\Phi = \exists X_{a_1} \ldots \exists X_{a_m} \exists X_{a'_1} \ldots \exists X_{a'_m} \left(\text{partition} \wedge \bigwedge_{a \in \Gamma'} \Psi_a \wedge \Psi_{\prec} \wedge \|\alpha'\| \right)$$

Roughly speaking, the formula partition (whose free variables are omitted here) guarantees that the basic MSC at hand is partitioned into sets represented by $X_{a_1}, \ldots, X_{a_m}, X_{a'_1}, \ldots, X_{a'_m}$. Intuitively, X_{a_i} and $X_{a'_i}$ decompose into all the prime partial MSCs that are single events in the trace representation of the MSC at hand. Thus, a formula Ψ_a ensures that, in turn, the set X_a can be partitioned into sets of events that each correspond to some event of a trace, which is a prime partial MSC. This can be done by a first-order formula. For example, taking $a \in \Gamma'$ to be the prime partial MSC G from Figure 3.9 on page 52 (recall that a_i and a'_i still both have to represent the same prime partial MSC), Ψ_a has to formalize that, for any x, $x \in X_a$ implies

- x is either labeled with 1!2 or with 2?1 and
- if x is labeled with 1!2, then there is an event $y \in X_a$ with $x \lhd_{\mathrm{c}} y$ such that any $z \notin \{x, y\}$ that is related to either x or y wrt. $\lhd$ is not contained in X_a and
- if x is labeled with 2?1, then there is an event $y \in X_a$ with $y \lhd_{\mathrm{c}} x$ such that any $z \notin \{x, y\}$ that is related to either x or y wrt. $\lhd$ is not contained in X_a.

Decomposing each set X_a into disjoint sets of events yields a refined partitioning of the MSC at hand, say into sets $X_1, \ldots, X_K$. Of course, K can be arbitrarily large. However, one can capture an element from $\mathcal{X} = \{X_1, \ldots, X_K\}$ by means of an $\mathrm{MSO}(Act, P_{\mathrm{c}})$-formula $\psi(X)$. Let us define an order $\prec$ on $\mathcal{X}$ according to $X_i \prec X_j$ if there are $x \in X_i$ and $y \in X_j$ both located on some process p such that $x \lhd_p y$. Now, $\Psi_\prec$ has to ensure that the reflexive transitive closure of $\prec$ is a well-defined partial order, which can be formalized by means of the (MSO-definable) transitive closure of the predicate $\|x \lhd y\|$ as specified below. Intuitively, this excludes an overlapping of prime partial MSCs. Moreover, $\|\alpha'\|$ is inductively derived from α' where

- $\|\lambda(x) = a\| = x \in X_a$,
- $\|x \in X\| = x \in X$,
- $\|x \lhd y\|$ formalizes that x and y respectively belong to some distinct sets $X_i, X_j \in \mathcal{X}$ with $X_i \subseteq X_a$ and $X_j \subseteq X_b$ for some a and b such that both $X_i \prec X_j$ and there is no set $X_k \in \mathcal{X}$ with $X_i \prec X_k \prec^+ X_j$. In particular, there must be a path from x to y via elements from $\lhd \cup \lhd^{-1}$ whose prefix consists of elements from X_a and which then only consists of elements from X_b where the transition from X_a to X_b must come from some $\lhd_p$ (which can be done by a first-order formula, as the length of a shortest path is restricted by a constant that only depends on Γ), and

– $\|x = y\|$ is defined similarly to $\|x \lhd y\|$ and formalizes that x and y belong to the same set X_a so that there is a path from x to y via elements from $\lhd \cup \lhd^{-1}$ that consists of elements from X_a only.

The other operators are derived canonically. □

It remains to identify a large subset of implementable gc-HcMSCs including some of those generating an (even existentially) unbounded behavior. In [Gen04, GKM04], it is already shown that any safe gc-HcMSC gives rise to an implementable MSC language, supplementing Theorem 5.1.3.

Theorem 5.5.3 ([Gen04, GKM04]) For any $\exists$-bounded MSC language L, $L \in$ gc-$\mathcal{H}$c$\mathcal{MSC}$ implies $L \in \mathcal{MSO}_{\mathbb{MSC}}$.

Actually, [Gen04, GKM04] even shows that an $\exists$-bounded MSC language is $\mathrm{MSO}_{\mathbb{MSC}}$-definable iff it is the MSC language of some safe HcMSC in which iteration occurs only over partial MSCs with connected communication graph.

Some results of this and the previous chapter are summarized in Figure 5.6 on page 119.

Chapter 6

Conclusion and Future Work

In this thesis, it is shown that MPAs are expressively equivalent to EMSO logic over MSCs. In particular, any EMSO sentence admits some implementation in terms of a finite MPA. Our proof is based on results by Thomas, which, in turn, refer to Hanf's Theorem. Moreover, we proved the class of MSO-definable MSC languages to be strictly larger than the class of implementable ones, concluding that MPAs cannot be complemented in general. This question was raised in [Kus01, Kus03]. Recall that we can also infer that the deterministic model of a finite MPA is strictly weaker than the nondeterministic one. In this regard, our main results read as follows:

Theorem 4.4.1 $\mathcal{MPA} = \mathcal{EMSO}_{\mathbb{MSC}}$

Theorem 5.2.1 The monadic quantifier-alternation hierarchy over $\mathbb{MSC}$ is infinite.

Theorem 5.2.6 $\mathcal{MPA}$ is not closed under complementation.

Theorem 5.3.2 $\mathcal{L}(\text{det-MPA}^f) \subsetneqq \mathcal{L}(\text{MPA}^f)$

We then showed that the specification formalism of HcMSCs is not complete in the sense that any implementable behavior can be specified. Conversely, however, the language of some gc-HcMSC turned out to be MSO-definable. This might be the starting point to finding large subsets of gc-$\mathcal{H}\text{c}\mathcal{MSC}$ that contain implementable MSC languages. In particular, it remains unanswered whether any gc-HcMSC is implementable.

Recall that, if we restrict to universally-bounded MSC languages, EMSO, MSO, EMSO[$\leq$], and MSO[$\leq$] coincide wrt. expressiveness. Thus, our work extends the work by Henriksen et al. [HMKT00b]. As a $\forall$-bounded MSC language can be seen as a Mazurkiewicz-trace language [Kus03] and, in the setting of traces, MSO logic is expressively equivalent to asynchronous automata, Theorem 4.4.1 might be understood as an extension of Zielonka's Theorem.

Let us once again discuss the role that the modeling of an MSC plays in this context. We consider an MSC to be a graph, which corresponds to the view taken in [LL95, Mad01] but is different from the one in [HMKT00b, Kus03], who model an MSC as a labeled partial order $(E, \leq, \lambda)$. However, while the way to define an MSC immediately affects the syntax and expressivity of (fragments of) the corresponding MSO logic, Theorems 4.4.1, 5.2.6, and 5.3.2 hold independently of that modeling, for the following reason: there is a one-to-one correspondence between an MSC structure $(E, \{\lhd_p\}_{p \in P}, \lhd_c, \lambda)$ and its counterpart $(E, \leq, \lambda)$ with $\leq\ = (\lhd_c \cup \bigcup_{p \in P} \lhd_p)^*$. As the definition of (a run of) an MPA is robust against the concrete modeling, too, Theorems 5.2.6 and 5.3.2 can be applied to any common definition of what an MSC is. However, our logic can only be considered to be the canonical (existential) MSO logic if MSCs are given by their graphs. We would like to stress that, actually, our main results are not restricted to the FIFO setting either and can be applied to MSCs with non-FIFO behavior in conjunction with message contents, too.

In the following, we list several open problems of interest and give suggestions what to do next:

- It remains open whether there is an EMSO[$\leq$]-sentence (or even FO[$\leq$]-sentence) whose language is not in $\mathcal{MPA}$. Note that we proved the corresponding result for MSO[$\leq$] only. Our conjecture is that there exists such a sentence. It is also an open problem what the exact relation between $\mathcal{EMSO}[\leq, \lhd_c]$ and $\mathcal{MPA}$ is.

- A further step might be to extend our results to infinite MSCs. In [Kus03], Kuske extends the finite setting to the infinite one in terms of regular MSC languages. As Hanf's Theorem has a counterpart for infinite graphs, it might be possible to obtain similar results.

- Our hierarchy of MPAs is primarily a hierarchy of *weakly* realizable MSC languages [AEY00], as their implementation is not necessarily free from deadlocks. It would be desirable to study deadlock-free MPAs in more detail, which give rise to *safely* realizable MSC languages. In particular, it would be worthwhile to study formalisms and logics that are implementable in terms of a safe and deterministic MPA.

- In [Gen04], it is shown that any safe gc-HcMSC is implementable. It remains to identify even larger sets of implementable HcMSCs. Recall that the existence of a defining EMSO-sentence is a sufficient criterion for implementability.

- In the context of traces, several temporal logics and their relation and complexity have been studied [Thi95, Wal98, DG00, Leu02]. In the framework of MSCs, similar studies are just at the beginning [MR00, MR04]. Of particular interest will be to find a canonical linear-time temporal logic, for example in the sense of Kamp's Theorem [Kam68, DG00].

Bibliography

[ABP97] A. Ayari, D. Basin, and A. Podelski. LISA: A specification language based on WS2S. In *Proceedings of Computer Science Logic, 12th International Workshop, CSL '98, Annual Conference of the EACSL, Brno, Czech Republic*, volume 1414 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 1997.

[AEY00] R. Alur, K. Etessami, and M. Yannakakis. Inference of Message Sequence Charts. In *Proceedings of the 22nd International Conference on Software Engineering*. ACM, 2000.

[AEY01] R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), Crete, Greece*, volume 2076 of *Lecture Notes in Computer Science*. Springer, 2001.

[Ara98] J. Araújo. Formalizing sequence diagrams. In L. Andrade, A. Moreira, A. Deshpande, and S. Kent, editors, *Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.

[AY99] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR 1999), Eindhoven, The Netherlands*, volume 1664 of *Lecture Notes in Computer Science*. Springer, 1999.

[BAL97] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Proceedings of the 3rd International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1997), Enschede, The Netherlands*, volume Lecture Notes in Computer Science 1217, pages 259–274. Springer, 1997.

[BB89] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. In P. H. J. van Eijk, C. A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 23–73. Elsevier Science Publishers North-Holland, 1989.

[BL01] B. Bollig and M. Leucker. Modelling, specifying, and verifying message passing systems. In Claudio Bettini and Angelo Montanari, editors, *Proceedings of the 8th International Symposium on Temporal Representation and Reasoning (TIME 2001), Civdale del Friuli, Italy*, pages 240–247. IEEE Computer Society Press, 2001.

[BL04] B. Bollig and M. Leucker. Message-Passing Automata are expressively equivalent to EMSO Logic. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR 2004), London, UK*, volume 3170 of *Lecture Notes in Computer Science*. Springer, 2004.

[BL05] B. Bollig and M. Leucker. Message-Passing Automata are expressively equivalent to EMSO Logic. *Theoretical Computer Science*, 2005. to appear.

[BLL02] B. Bollig, M. Leucker, and P. Lucas. Extending Compositional Message Sequence Graphs. In *Proceedings of the 9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2002), Tbilisi, Georgia*, volume 2514 of *Lecture Notes in Computer Science*. Springer, 2002.

[BLN02] B. Bollig, M. Leucker, and T. Noll. Generalised regular MSC languages. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2002), Grenoble, France*, volume 2303 of *Lecture Notes in Computer Science*. Springer, 2002.

[Blu01] Specification of the Bluetooth System (version 1.1), 2001. http://www.bluetooth.com.

[BM03] N. Baudru and R. Morin. Safe implementability of regular message sequence chart specifications. In *Proceedings of the ACIS Fourth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2003), Lübeck, Germany*, volume 2380 of *Lecture Notes in Computer Science*. Springer, 2003.

[BM04] N. Baudru and R. Morin. The pros and cons of netcharts. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR 2004), London, UK*, volume 3170 of *Lecture Notes in Computer Science*. Springer, 2004.

[Büc60] J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.

[BV94] J. C. M. Baeten and C. Verhoef. Concrete process algebra. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 149–268. Oxford University Press, 1994.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.

[CE81] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CGP99] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[Cou90] B. Courcelle. The monadic second order logic of graphs I: recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[DG00] V. Diekert and P. Gastin. LTL is expressively complete for Mazurkiewicz traces. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP 2000), Geneva, Switzerland*, volume 1853 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2000.

[DGK00] M. Droste, P. Gastin, and D. Kuske. Asynchronous cellular automata for pomsets. *Theoretical Computer Science*, 247(1-2), 2000.

[DH01] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19:1:45–80., 2001.

[DR95] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

[Ebi95] Werner Ebinger. Logical definability of trace languages. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 10, pages 382–390. World Scientific, Singapore, 1995.

[Elg61] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[Gen04] B. Genest. *L'Odyssée des Graphes de Diagrammes de Séquences (MSC-Graphes)*. PhD thesis, Laboratoire d'Informatique Algorithmique: Fondements et Applications (LIAFA), 2004.

[GKM04] B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem for a class of communicating automata with effective algorithms. In *Proceedings of the 8th International Conference on Developments in Language Theory (DLT 2004), Auckland, New Zealand*, volume 3340 of *Lecture Notes in Computer Science*. Springer, 2004.

[GMMP04] B. Genest, M. Minea, A. Muscholl, and D. Peled. Specifying and verifying partial order properties using template MSCs. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2004), Barcelona, Spain*. Lecture Notes in Computer Science, 2004.

[GMP01] E. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001), Genova, Italy*, volume 2031 of *Lecture Notes in Computer Science*. Springer, 2001.

[GMSZ02] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state high-level MSCs: Model-checking and realizability. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP 2002), Malaga, Spain*, volume 2380 of *Lecture Notes in Computer Science*. Springer, 2002.

[GRST96] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Information and Computation*, 125(1):32 – 45, 1996.

[GTW02] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[Han65] W. P. Hanf. Model-theoretic methods in the study of elementary logic. In J. W. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*. North-Holland, Amsterdam, 1965.

[HJJ+95] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Proceedings of the First International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1995), Aarhus, Denmark*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110. Springer, 1995.

[HK02] D. Harel and H. Kugler. Synthesizing state-based object systems from LSCs specifications. *Foundations of Computer Science*, 13:1:5–51, 2002.

[HM00] L. Hélouët and P. Le Magait. Decomposition of message sequence charts. In *Proceedings of SAM 2000, 2nd Workshop on SDL and MSC*. VERIMAG, IRISA, SDL Forum, 2000.

[HMK+05] J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 2005. to appear.

[HMKT99] J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. Towards a theory of regular MSC languages. Technical Report RS-99-52, Department of Computer Science, University of Aarhus, 1999.

[HMKT00a] J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP 2000), Geneva, Switzerland*, volume 1853 of *Lecture Notes in Computer Science*. Springer, 2000.

[HMKT00b] J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. Regular collections of message sequence charts. In *Proceedings of the 25th International Symposium Mathematical Foundations of Computer Science (MFCS 2000), Bratislava, Slovakia*, volume 1893 of *Lecture Notes in Computer Science*. Springer, 2000.

[ITU98] ITU-TS Recommendation Z.120anb: Formal Semantics of Message Sequence Charts, 1998.

[ITU99] ITU-TS Recommendation Z.120: Message Sequence Chart 1999 (MSC99), 1999.

[Kam68] H. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[KL98] J.-P. Katoen and L. Lambert. Pomsets for message sequence charts. In H. König and P. Langendörfer, editors, *Formale Beschreibungstechniken für Verteilte Systeme*, pages 197–208, Cottbus, Germany, 1998. Shaker Verlag.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies, Annals of Math. Studies 34*, pages 3–40. Princeton, New Jersey, 1956.

[Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, December 1983.

[Kus01] D. Kuske. Another step towards a theory of regular MSC languages. Technical Report 36, Department of Mathematics and Computer Science, University of Leicester, 2001.

[Kus02] D. Kuske. A further step towards a theory of regular MSC languages. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2002), Antibes - Juan les Pins, France*, volume 2285 of *Lecture Notes in Computer Science*. Springer, 2002.

[Kus03] D. Kuske. Regular Sets of Infinite Message Sequence Charts. *Information and Computation*, 187:80–109, 2003.

[Leu02] M. Leucker. *Logics for Mazurkiewicz traces*. PhD thesis, Lehrstuhl für Informatik II, RWTH Aachen, 2002.

[LL95] P. B. Ladkin and S. Leue. Interpreting message flow graphs. *Formal Aspects of Computing*, 7(5):473–509, 1995.

[LM04] M. Lohrey and A. Muscholl. Bounded MSC Communication. *Information and Computation*, 189(2):160–181, 2004.

[LMM02] M. Leucker, P. Madhusudan, and S. Mukhopadhyay. Dynamic message sequence charts. In *Proceedings of the 22nd Conference on*

Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2002), Kanpur, India, volume 2556 of *Lecture Notes in Computer Science*. Springer, 2002.

[Loh03] M. Lohrey. Realizability of high-level message sequence charts: closing the gaps. *Theoretical Computer Science*, 309(1-3):529–554, 2003.

[Mad01] P. Madhusudan. Reasoning about Sequential and Branching Behaviours of Message Sequence Graphs. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), Crete, Greece*, volume 2076 of *Lecture Notes in Computer Science*. Springer, 2001.

[Mau96] S. Mauw. The formalization of message sequence charts. *Computer Networks and ISDN Systems*, 28(12):1643-1657, 1996.

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[MKRS98] M. Mukund, K. Narayan Kumar, J. Radhakrishnan, and M. Sohoni. Towards a characterisation of finite-state message-passing systems. In *Proceedings of Advances in Computing Science (ASIAN 1998), 4th Asian Computing Science Conference, Manila, The Philippines*, volume 1538 of *Lecture Notes in Computer Science*. Springer, 1998.

[MKS00] M. Mukund, K. Narayan Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000), University Park, PA, USA*, volume 1877 of *Lecture Notes in Computer Science*. Springer, 2000.

[MKT03] M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. Netcharts: Bridging the gap between HMSCs and executable specifications. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR 2003), Marseille, France*, volume 2761 of *Lecture Notes in Computer Science*, pages 296–30. Springer, 2003.

[MM01] P. Madhusudan and B. Meenakshi. Beyond message sequence graphs. In *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2001), Bangalore, India*, volume 2245 of *Lecture Notes in Computer Science*. Springer, 2001.

[Mor01] R. Morin. On regular message sequence chart languages and relationships to Mazurkiewicz trace theory. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2002), Grenoble, France*, volume 2030 of *Lecture Notes in Computer Science*. Springer, 2001.

[Mor02] R. Morin. Recognizable sets of message sequence charts. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2002), Antibes – Juan les Pins, France*, volume 2285 of *Lecture Notes in Computer Science*. Springer, 2002.

[MP99] A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science (MFCS 1999), Szklarska Poreba, Poland*, volume 1672 of *Lecture Notes in Computer Science*. Springer, 1999.

[MP01] A. Muscholl and D. Peled. From finite state communication protocols to high-level message sequence charts. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), Crete, Greece*, volume 2076 of *Lecture Notes in Computer Science*. Springer, 2001.

[MPS98] A. Muscholl, D. Peled, and Z. Su. Deciding properties for message sequence charts. In *Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structures (FOSSACS 1998), Lisbon, Portugal*, volume 1578 of *Lecture Notes in Computer Science*. Springer, 1998.

[MR97] S. Mauw and M. A. Reniers. High-level message sequence charts. In *Proceedings of the Eighth SDL Forum (SDL'97)*, pages 291–306, 1997.

[MR00] B. Meenakshi and R. Ramanujam. Reasoning about message passing in finite state environments. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP 2000), Geneva, Switzerland*, volume 1853 of *Lecture Notes in Computer Science*. Springer, 2000.

[MR04] B. Meenakshi and R. Ramanujam. Reasoning about layered message passing systems. *Computer Languages, Systems, and Structures*, 30(3-4):529–554, 2004.

[MST02] O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. *Information and Computation*, 179(2), 2002.

[MT97] O. Matz and W. Thomas. The monadic quantifier alternation hierarchy over graphs is infinite. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS 1997), Warsaw, Poland*. IEEE Computer Society Press, 1997.

[Och95] E. Ochmański. Recognizable Trace Languages. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 6, pages 167–204. World Scientific, Singapore, 1995.

[Per91] D. Perry. *VHDL*. McGraw-Hill, New York, 1991.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS 1977)*, pages 46–57, Providence, Rhode Island, 1977. IEEE Computer Society Press.

[RV01] A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier, 2001.

[SP99] P. Stevens and R. Pooley. *Using UML: software engineering with objects and components*. Object Technology Series. Addison-Wesley Longman, 1999.

[Thi95] P. S. Thiagarajan. A trace consistent subset of PTL. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR 1995), Philadelphia, PA, USA*, volume 962 of *Lecture Notes in Computer Science*. Springer, 1995.

[Tho90] W. Thomas. On logical definability of trace languages. In V. Diekert, editor, *Proceedings of a workshop of the ESPRIT Basic Research Action No 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS), Kochel am See, Bavaria, FRG (1989)*, Report TUM-I9002, Technical University of Munich, pages 172–182, 1990.

[Tho91] W. Thomas. On Logics, Tilings, and Automata. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming (ICALP 1991), Madrid, Spain*, volume 510 of *Lecture Notes in Computer Science*. Springer, 1991.

[Tho96] W. Thomas. Elements of an automata theory over partial orders. In *Proceedings of Workshop on Partial Order Methods in Verification (POMIV 1996)*, volume 29 of *DIMACS*. AMS, 1996.

[Tho97a] W. Thomas. Automata theory on trees and partial orders. In *Proceedings of TAPSOFT 1997: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Lille, France*, volume 1214 of *Lecture Notes in Computer Science*. Springer, 1997.

[Tho97b] W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words. Springer, Berlin, 1997.

[Wal98] I. Walukiewicz. Difficult configurations – on the complexity of LTrL. In *Proceedings of 25th International Colloquium on Automata, Languages and Programming (ICALP 1998), Aalborg, Denmark*, volume 1443 of *Lecture Notes in Computer Science*, pages 140–151, 1998.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.

Appendix A

The Counterexample

We provide here the complete counterexample for the simpler transformation from a graph acceptor into a finite MPA on page 93.

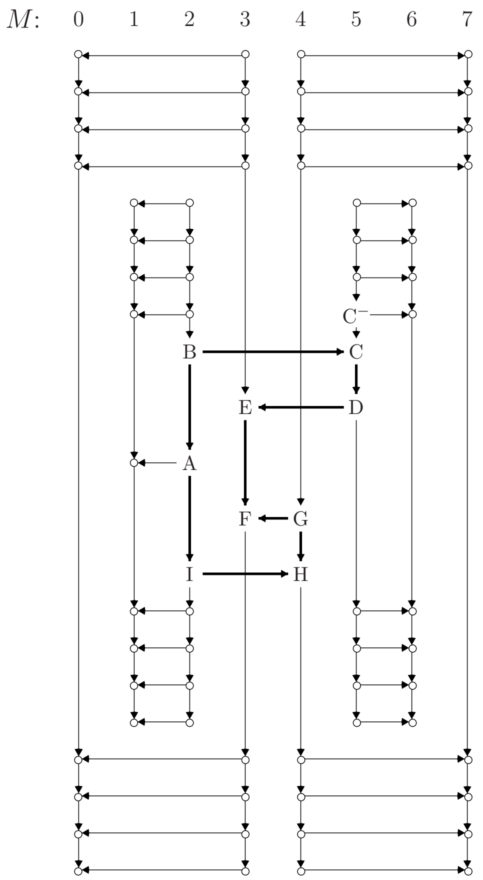


Figure A.1: Counterexample for the simpler transformation

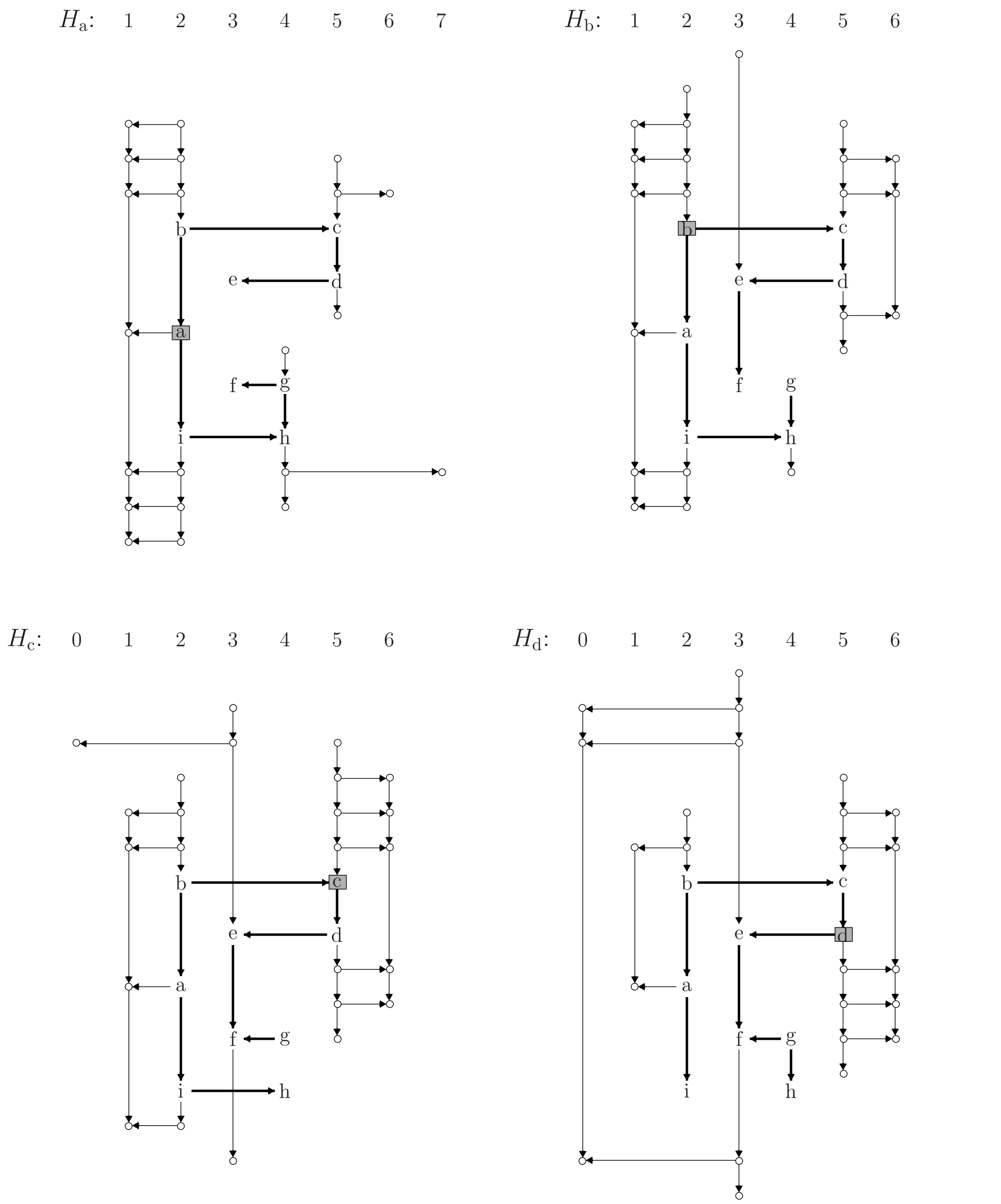


Figure A.2: Counterexample for the simpler transformation

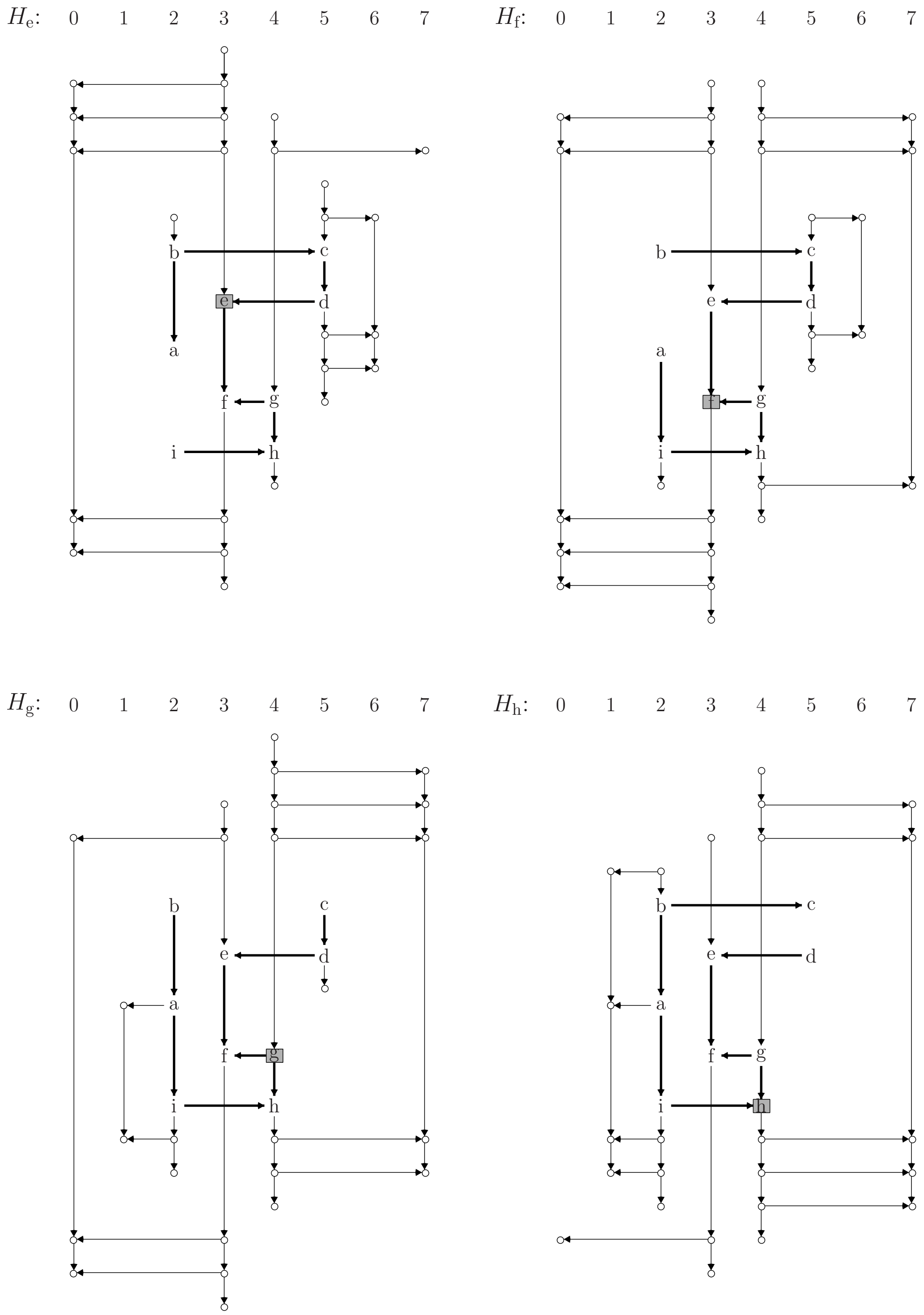


Figure A.3: Counterexample for the simpler transformation

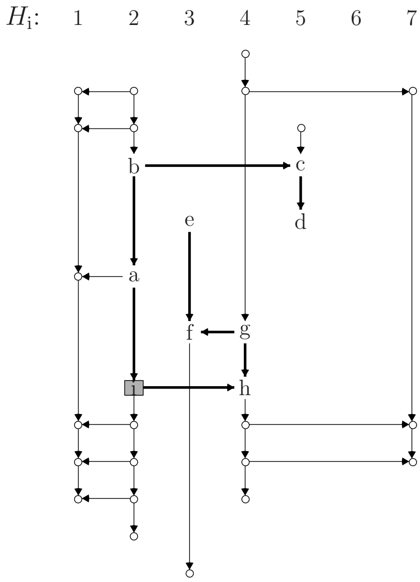


Figure A.4: Counterexample for the simpler transformation

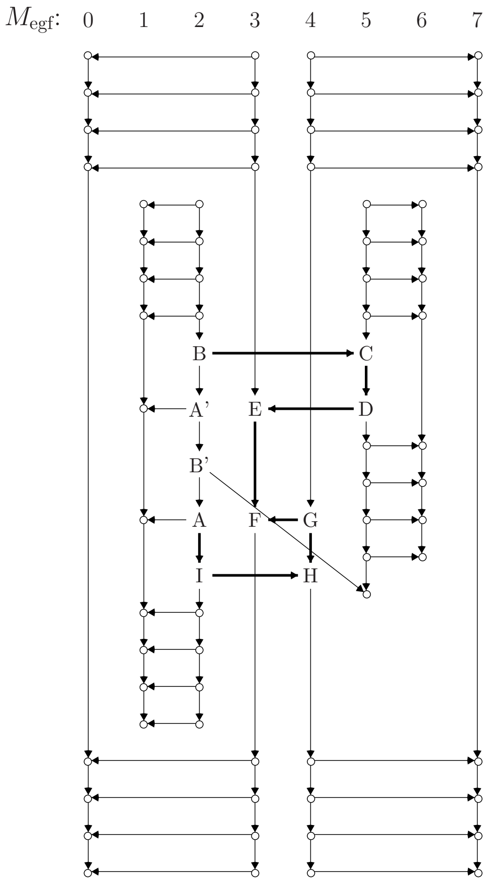


Figure A.5: A witness for $(H_e, 3?4, H_g, H_f)$

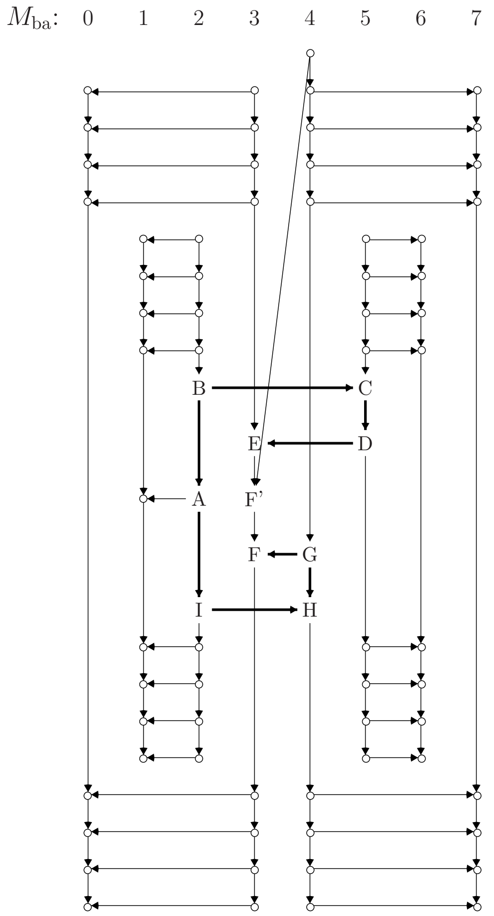


Figure A.6: A witness for $(H_b, 2!1, H_a, H_a)$

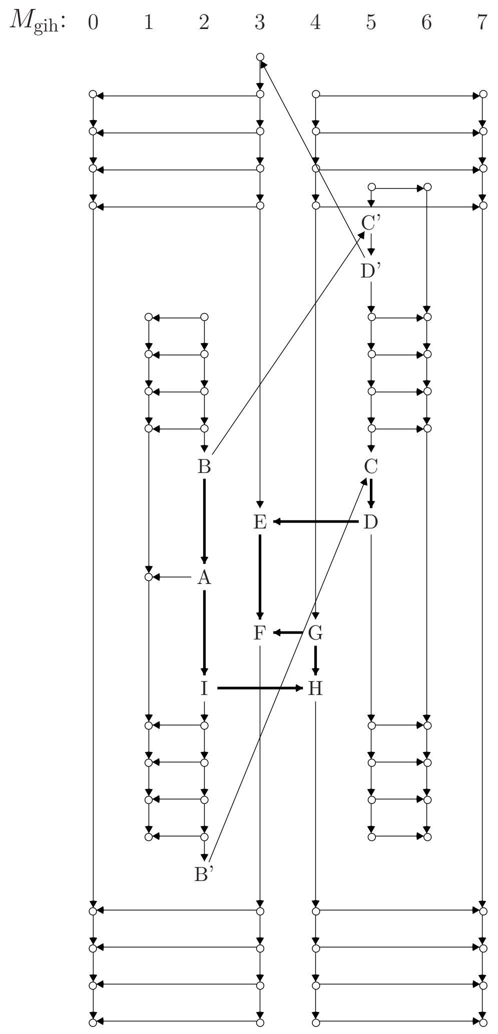


Figure A.7: A witness for $(H_{\mathrm{g}}, 4?2, H_{\mathrm{i}}, H_{\mathrm{h}})$

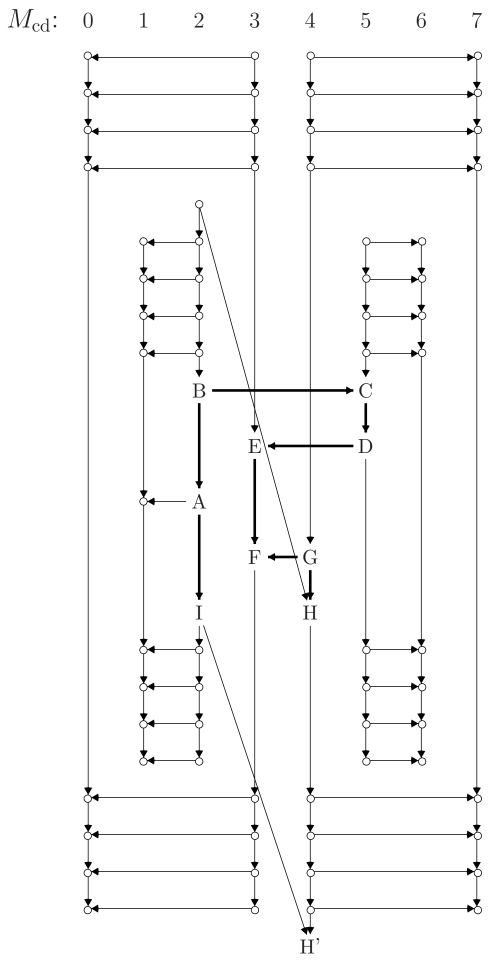


Figure A.8: A witness for $(H_{\mathrm{c}}, 5!3, H_{\mathrm{d}}, H_{\mathrm{d}})$

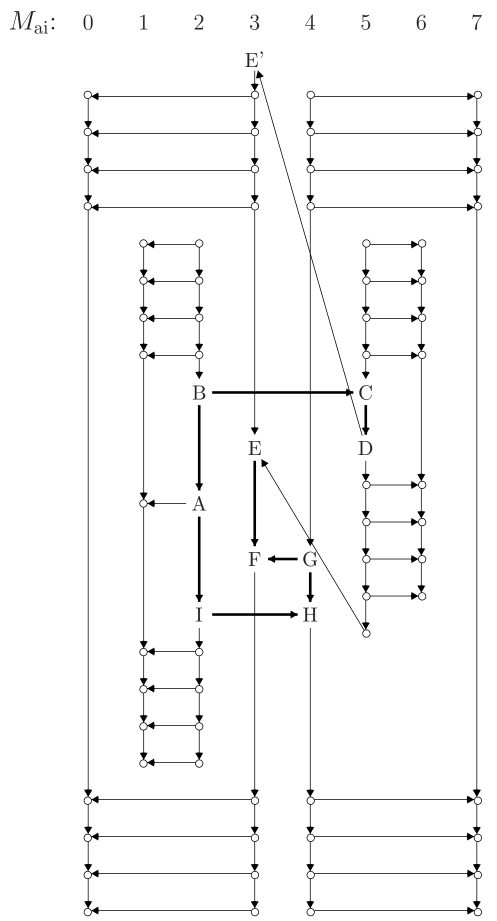


Figure A.9: A witness for $(H_{\mathrm{a}}, 2!4, H_{\mathrm{i}}, H_{\mathrm{i}})$

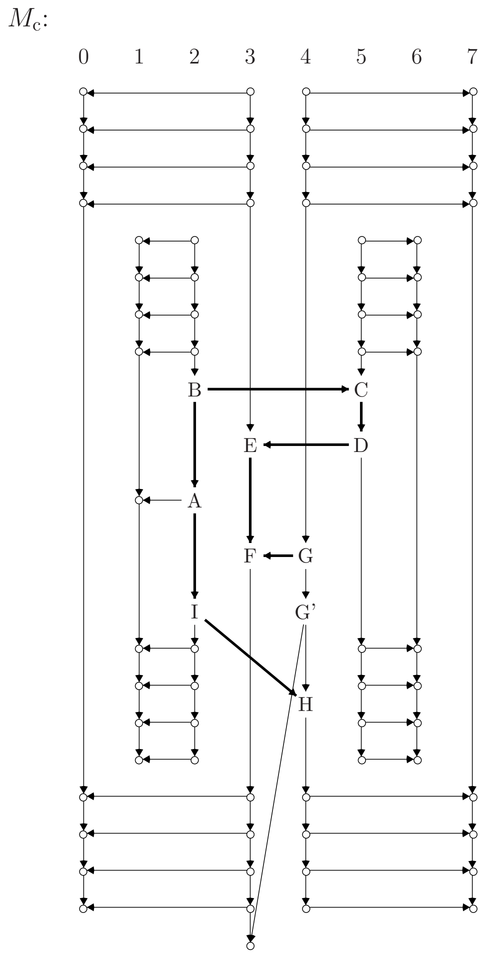


Figure A.10: A witness for $(4\text{-Sph}(M, \mathrm{C}^-), 5?2, H_\mathrm{b}, H_\mathrm{c})$

Appendix B

Symbols and Notations

$\mathcal{RAT}_{\mathbb{M}}$	rational subsets of $\mathbb{M}$, 11
$\mathcal{REC}_{\mathbb{M}}$	recognizable subsets of $\mathbb{M}$, 12
$\lhd$	edge relation, 12
$\lvert G\rvert$	cardinality of G, 12
$\mathbb{DG}$	set of graphs, 13
$\mathbb{DG}_H$	set of graphs, 13
$G\restriction\Sigma'$	projection, 13
$\lambda_{\lvert E'}$	restriction of λ to E', 13
$G\Downarrow e$	downwards closure, 14
$G\downarrow e$	strict downwards closure, 14
$\mathcal{K}^Q$	Q-extensions of $\mathcal{K}$, 14
MSO	monadic second-order formulas, 15
FO	first-order formulas, 16
EMSO	existential monadic second-order formulas, 16
Σ_k	Σ_k-formulas, 16
$\mathcal{MSO}_{\mathcal{K}}$	MSO-definable languages, 17
$\mathcal{FO}_{\mathcal{K}}$	FO-definable languages, 17
$\mathcal{EMSO}_{\mathcal{K}}$	EMSO-definable languages, 17
$\mathcal{L}_{\mathcal{K}}(\Sigma_k)$	Σ_k-definable languages, 17
$\mathcal{LTT}_{\mathcal{K}}$	locally threshold testable languages, 19
$d_G(e',e)$	distance from e' to e, 19
R-Sph(G,e)	R-sphere of G around e, 19
$L_{\mathcal{K}}(\mathcal{B})$	language of $\mathcal{B}$ relative to $\mathcal{K}$, 21
$\mathcal{GA}_{\mathcal{K}}$	languages of graph acceptors, 21
k-$\mathcal{GA}_{\mathcal{K}}$	languages of graph acceptors with k-spheres, 21
$\mathbb{W}$	words, 22
$\mathcal{FA}$	languages of finite automata, 24

$Lin(G)$	linearizations of G, 25
$loc(a)$	agents involved in a, 26
$\tilde{\Sigma}$	distributed alphabet, 26
$\mathbf{1}_{\mathbb{TR}}$	unit trace, 27
c-$\mathcal{RAT}_{\mathbb{TR}}$	c-rational subsets of $\mathbb{TR}$, 28
$\mathcal{R}_{\mathbb{TR}}$	regular trace languages, 29
$L \vdash_{\tilde{\Sigma}} T$	trace inference, 29
$\mathcal{P}^0_{\mathbb{TR}}$	weak product trace languages, 29
$\mathcal{P}_{\mathbb{TR}}$	product trace languages, 29
$\mathcal{RP}^0_{\mathbb{TR}}$	weak regular product trace languages, 29
$\mathcal{RP}_{\mathbb{TR}}$	regular product trace languages, 29
$\mathcal{AA}$	languages of asynchronous automata, 31
$\mathcal{PA}$	languages of product automata, 33
$[n]$	$\{1, \ldots, n\}$, 34
$\mathbb{P}$	pictures, 34
$\mathbb{GR}$	grids, 36
P	processes, 39
Ch	channels, 39
$Act^!$	send actions, 39
$Act^?$	receive actions, 39
Act	actions, 39
Com	communicating actions, 39
U_M	unmatched events of M, 40
p$\mathbb{MSC}$	partial MSCs, 41
$\mathbb{MSC}$	MSCs, 41
$\mathcal{MSC}$	MSC languages, 41
$\mathbf{1}_{\mathbb{MSC}}$	unit MSC, 43
$P(M)$	processes of M, 43
c$G(M)$	communication graph of M, 43
$\mathbb{MSC}_{\forall B}$	universally-B-bounded MSCs, 44
$\mathbb{MSC}_{\exists B}$	existentially-B-bounded MSCs, 44
$\mathfrak{L}(\mathcal{H})$	MSC language of $\mathcal{H}$, 50
$\mathcal{H}$c$\mathcal{MSC}$	MSC languages of HcMSCs, 53
gc-$\mathcal{H}$c$\mathcal{MSC}$	MSC languages of gc-HcMSCs, 53
safe-gc-$\mathcal{H}$c$\mathcal{MSC}$	MSC languages of safe gc-HcMSCs, 53
left-closed-gc-$\mathcal{H}$c$\mathcal{MSC}$	MSC languages of left-closed gc-HcMSCs, 53
$\mathcal{HMSC}$	MSC languages of HMSCs, 53
gc-$\mathcal{HMSC}$	MSC languages of gc-HMSCs, 53

Index

Aachener Informatik-Berichte

This is a list of recent technical reports. To obtain copies of technical reports please consult `http://aib.informatik.rwth-aachen.de/` or send your request to: `biblio@informatik.rwth-aachen.de`

2000-01 *	Jahresbericht 1999
2000-02	Jens Vöge, Marcin Jurdzinski: A Discrete Strategy Improvement Algorithm for Solving Parity Games
2000-04	Andreas Becks, Stefan Sklorz, Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach
2000-05	Mareike Schoop: Cooperative Document Management
2000-06	Mareike Schoop, Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling
2000-07 *	Markus Mohnen, Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages
2000-08	Thomas Arts, Thomas Noll: Verifying Generic Erlang Client-Server Implementations
2001-01 *	Jahresbericht 2000
2001-02	Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03	Thierry Cachat: The power of one-letter rational languages
2001-04	Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free μ-Calculus
2001-05	Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06	Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07	Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08	Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09	Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10	Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11	Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 *	Jahresbericht 2001

2002-02	Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03	Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04	Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05	Horst Lichter, Thomas von der Maen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06	Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07	Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08	Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09	Markus Mohnen: Interfaces with Default Implementations in Java
2002-10	Martin Leucker: Logics for Mazurkiewicz traces
2002-11	Jürgen Giesl, Hans Zantema: Liveness in Rewriting
2003-01 *	Jahresbericht 2002
2003-02	Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting
2003-03	Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations
2003-04	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs
2003-05	Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06	Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates
2003-07	Horst Lichter, Thomas von der Maen, Alexander Nyen, Thomas Weiler: Vergleich von Anstzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung
2003-08	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs
2004-01 *	Fachgruppe Informatik: Jahresbericht 2003
2004-02	Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03	Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04	Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming

2004-05	Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06	Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07	Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08	Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09	Joachim Kneis, Daniel Mlle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10	Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 *	Fachgruppe Informatik: Jahresbericht 2004
2005-02	Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"
2005-03	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04	Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05	Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06	Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07	Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08	Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09	Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures

* These reports are only available as a printed version.

Curriculum Vitae

Name	Benedikt Michael Peter Bollig
Geburtsdatum	13.01.1975
Geburtsort	Düsseldorf

Bildungsgang

1981 – 1985	Pestalozzi-Grundschule Neuss
1985 – 1994	Quirinus-Gymnasium Neuss Abschluss: Allgemeine Hochschulreife
1994 – 2000	Studium der Informatik an der RWTH Aachen Abschluss: Diplom
seit 2000	Wissenschaftlicher Angestellter am Lehrstuhl für Informatik II (Prof. Dr. Klaus Indermark), RWTH Aachen