Benedikt Bollig

Formal Models of Communicating Systems

Languages, Automata, and Monadic Second-Order Logic

Springer

Formal Models of Communicating Systems

Benedikt Bollig

Formal Models of Communicating Systems

Languages, Automata, and Monadic Second-Order Logic

With 72 Figures and 6 Tables

Springer

Author

Benedikt Bollig

Laboratoire Spécification et Vérification
CNRS UMR 8643 & ENS de Cachan
94235 Cachan Cedex
France
bollig@lsv.ens-cachan.fr

Library of Congress Control Number: 2006928323

ACM Computing Classification (1998): F.1, F.2, F.3, F.4, G.2.2.

ISBN-10 3-540-32922-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-32922-0 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typeset by the author
Production: LE-TEX Jelonek, Schmidt & Vöckler GbR, Leipzig
Cover design: KünkelLopka Werbeagentur, Heidelberg

Printed on acid-free paper 45/3100/YL - 5 4 3 2 1 0

Preface

The close connection between automata and logic has ever been a fascinating subject of theoretical computer science. The origins of that area go back to Büchi and Elgot, who showed at the beginning of the 60's that formulas from monadic second-order logic and finite automata have the same expressive power.

Since then, a large amount of research has been accomplished to extend those results to other settings such as infinite words, trees, traces, and grids. The benefits of precise characterizations of state-based, *operational* automata models in terms of *descriptive* logical formalisms are twofold. On the one hand, they allow us to derive algorithmic and logical properties of the model. On the other hand, from a software engineer's perspective, fragments of monadic second-order logic might be used to *specify* the desired system behavior, which is then reflected in an automata implementation.

This book studies the relation between automata and monadic second-order logic. In doing so, it focuses on classes of automata that describe the concurrent behavior of a distributed system. For example, we will bridge the gap between monadic second-order logic and channel systems, which communicate via reliable or faulty fifo ("first-in, first-out") queues. Moreover, we will study systems that synchronize when simultaneously accessing a common device. Due to the complexity of those communication paradigms, the formal treatment of related systems in terms of automata models and equivalent logical formalisms plays an important role in their synthesis and verification.

The reader is assumed to have only some basic knowledge in theoretical computer science (e.g., about finite automata and formal languages) and to be familiar with mathematical terminology and notation. Thus, the book should be accessible to senior undergraduate or graduate students. Please note that any relevant information in conjunction with this book, such as course material, solutions to selected exercises, list of errata, etc., will be provided on its homepage, which is located at

`http://www.lsv.ens-cachan.fr/~bollig/fmcs/`

A large part of this book is based on my Ph.D. thesis "Automata and Logics for Message Sequence Charts", which I wrote at RWTH Aachen University.

First and foremost, I wish to express my gratitude and admiration to my former supervisor, Prof. Klaus Indermark. His enthusiasm and continuous support, his advice and suggestions have influenced me greatly as a computer scientist and enabled me to write this book.

I am grateful to Prof. Wolfgang Thomas, without whose seminal scientific contributions and inspiring and insightful lectures the results documented in this book would not have been possible.

My special thanks go to 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 gratefully acknowledge the funding of my stay at Birmingham University by the German Academic Exchange Service (DAAD). I thank Prof. Marta Kwiatkowska for giving me the opportunity to work with her and her excellent group during that memorable time.

Parts of this book were written in Cachan, France, where I currently hold a CNRS research position at the Laboratoire Spécification et Vérification.

I feel very much indebted to Marie Duflot-Kremer and Steve Kremer. Thank you for all your hospitality, graciousness, and patience, which made me feel at home from the day I moved to France.

Many thanks go to everyone at the Laboratoire Spécification et Vérification for their warm welcome and support and for an outstanding, motivating, and friendly research environment.

I thank Springer-Verlag for editorial assistance and Ronan Nugent for his continuous support and helpfulness during the whole production process of the book.

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.

Cachan, France — Benedikt Bollig
April 2006

Contents

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 were unthinkable some years ago and their complexity is still rapidly growing so that it becomes more and more difficult to detect errors or to predict their incidence. Consequently, *formal methods* play an increasing 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, we are faced with 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 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 accompany 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 book are *communicating automata*, which, though they might abstract from many details, reflect the operational 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 must 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 (MSO) logic [8, 44], the temporal logics LTL [83] and CTL [22], and the μ-calculus [54]. A specification might also be given in a high-level language that is closer to an implementation and often allows us to *synthesize* a system directly and automatically. In this regard, let us mention some process-algebra based languages such as CCS [71], ACP [9], and LOTOS [18] and other formal design notions like VHDL [81]. In this book, 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. 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* [23], which is fully automatic, and *theorem proving* [85], which requires human assistance. If, otherwise, a system is synthesized directly from its specification, then it can be assumed to be correct a priori, provided the translation preserves the semantics of the specification.

Several phases of system design are depicted in Fig. 1.1, which, in addition, features the stage of *code generation* to gain from the system model an effective implementation thereof.

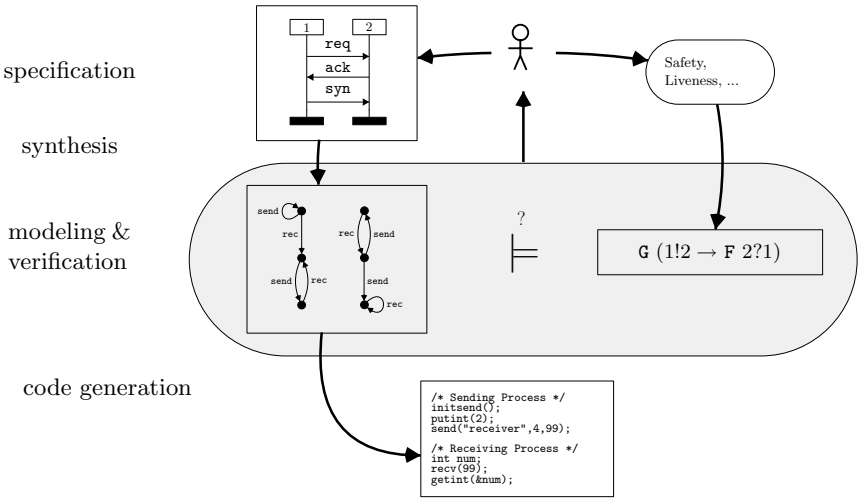
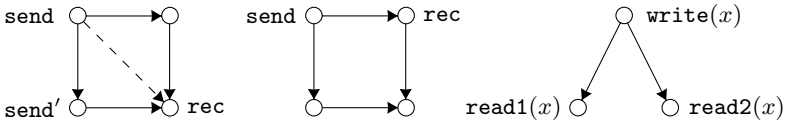


Fig. 1.1. Stages of system development

1.2 Partial Orders and Graphs

As mentioned above, it is desirable to apply formal methods even in the early stages of system design to avoid extensive reimplementation and redesign, which, in turn, might lead to explosive costs. A common design practice when developing communicating systems is to start with specifying scenarios to exemplify the intended interaction of the system to be. Usually, distributed systems operate concurrently, i.e., some actions do not depend on the occurrence of another. One possible single execution sequence of a distributed system is therefore often described by a partially ordered set (poset) $(V, \leq)$, such as depicted by Fig. 1.2a. The elements of V, which are also referred to as *events*, comprise actions that are executed during a system run. They are arranged according to the partial order $\leq \subseteq V \times V$ to reflect their interaction dependencies. Say, for example, we deal with events $\texttt{send}, \texttt{rec} \in V$ that form the send and receipt of a message. Naturally, sending a message precedes its receipt so that $\texttt{send} \leq \texttt{rec}$. Otherwise, there might be events that do not interfere with each other. For example, two read events $\texttt{read1}(x)$ and $\texttt{read2}(x)$ that independently read a shared variable x are not related, so neither $\texttt{read1}(x) \leq \texttt{read2}(x)$ nor $\texttt{read2}(x) \leq \texttt{read1}(x)$, whereas the time of writing the variable does affect the value of what is read (cf. Fig. 1.2c).

A poset, in turn, might be represented by a graph $(V, \rightarrow)$ whose edge relation $\rightarrow \subseteq V \times V$ gives rise to $\leq$ when generating its reflexive transitive closure. Sometimes, $\rightarrow$ allows a more concrete modeling of communication than $\leq$. Namely, writing $\texttt{send} \rightarrow \texttt{rec}$ suggests that send and rec together



(a) a poset (b) a graph (c) independence of events

Fig. 1.2. Partially ordered sets and graphs

form the exchange of *one and the same* message (cf. Fig. 1.2b), whereas writing `send` $\leq$ `rec` is actually a weaker statement, just claiming that `rec` happens eventually after `send` but might be the receipt of another event `send′`, as illustrated in Fig. 1.2a.

Message sequence charts (MSCs) provide a prominent notion to further the partial-order and graph-based approaches. They are widely used in industry, are standardized [49, 50], and are similar to UML's sequence diagrams [7]. An MSC depicts a single partially ordered execution sequence of a system. In doing so, it defines a collection of processes, which, in their 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* [13], a specification for wireless communication, is depicted in Fig. 1.3. 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 a `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 the form of a `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 depends on the visual arrangement of arrows. An endpoint of an arrow is a *send event* if it is the source of that arrow. Otherwise, it is a *receive event*. More specifically, we suppose events located on one and the same process line to be totally ordered and require a receive event to occur only if the corresponding send event has been executed. The above-mentioned 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, re-

ceiving `HCI_Command_Status` 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.

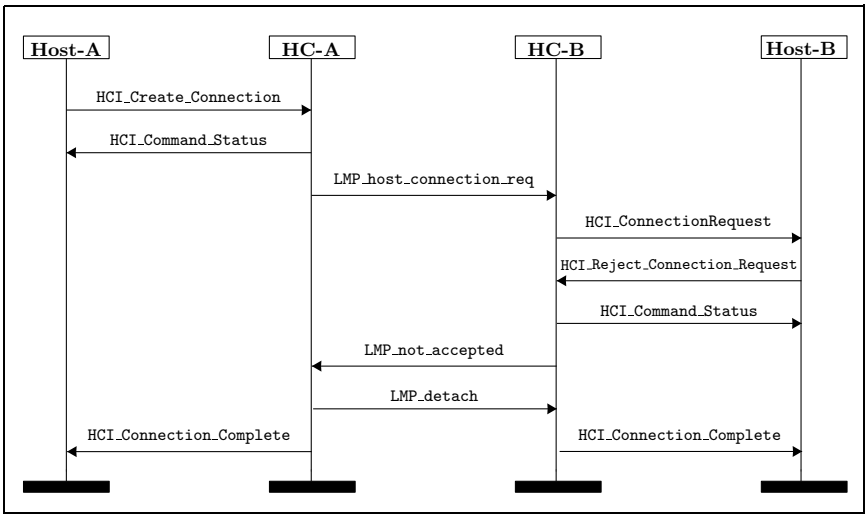


Fig. 1.3. An MSC modeling the ACL connection request phase

1.3 High-Level Specifications

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. A single graph or poset can, however, describe no more than one single execution sequence. Otherwise, a collection of graphs might capture all the scenarios that a designer wants the system under development to realize. Based on the notion of MSCs and the likewise partial-order based concept of *Mazurkiewicz traces* [27], several modeling and specification formalisms have been considered at a formal level, among them *high-level MSCs* [6, 45, 68, 76], 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 languages* by means of choice, concatenation, and iteration. The study of algebraic language classes might then lead us to *recognizable languages* [73, 96], which can be characterized by certain *monoid automata*. Following the classical algebraic approach further, we will come across the class of *regular languages* whose linear extensions form a regular word language.

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 [35, 56, 72, 73]. 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 poset- or graph-based counterpart of a finite automaton over words, which is explained in the next section in more detail.

MSO logic provides another specification formalism. But not only does MSO logic constitute an expressive specification language. Its relation to formalisms such as automata or high-level constructs over graphs and posets has also been a research area of great interest aiming at a deeper understanding of the latter's logical and algorithmic properties (see [94] for an overview). Following the logical approach, one might likewise argue that we can call a set of graphs regular if it is definable in the corresponding MSO logic, because, in the domain of words, regularity and definability in MSO logic coincide [20, 32].

1.4 Towards an Implementation

The next step in system design might be to supply an implementation realizing or satisfying a specification. Recall that we are still interested in an abstract model rather than a concrete implementation in some low-level programming language. However, the view we are taking now is much closer to the latter. More precisely, we ask for *automata* models that are suited to 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 above-mentioned language classes.

To create formal methods tailored to a distributed system and to the associated mathematical model, it is generally helpful to study some of the model's properties first and to learn more about its limitations along with algorithmic restrictions and its degree of abstraction. In this regard, typical questions to clarify are:

- Is my model of an implementation a suitable one, i.e., does it reflect all the aspects I want to verify?
- What is a suitable specification language; is any specification implementable?
- What kind of problem can I expect to be decidable?

Basically, that is what this book is all about. We will hereby concentrate on communicating systems, which occur whenever independent processes and objects interact, whether via message exchange through fifo ("first-in, first-out") buffers or when attempting to write a shared variable. At the same time, we focus on issues related to the areas of system modeling and specification.

In particular, we will address the relation of several automata models with (fragments of) MSO logic to clarify its use as a specification language.

Concerning systems that are distributed in nature, the notion of a process is central. It seems therefore natural to consider each process as a single automaton and to define a notion of communication describing how these parallel systems work together. When, for example, we equip such local processes with message buffers, we obtain the model of a *message-passing automaton* or *communicating (finite-state) machine*. There is a precise logical characterization of communicating finite-state machines by a fragment of MSO logic, called existential MSO (EMSO) logic, so that any specification in terms of an EMSO expression has an implementation in terms of a communicating finite-state machine. Another model of communication is provided by *asynchronous automata*. Herein, local processes synchronize by executing certain actions (e.g., writing a variable) simultaneously, whereas others may be taken autonomously (e.g., reading the variable). Asynchronous automata were introduced originally by Zielonka in the framework of the partial-order model of Mazurkiewicz traces [97], and they were generalized by Droste et al. to run on even more general posets [29]. Asynchronous automata could also be shown to be expressively equivalent to EMSO logic relative to traces and CROW-posets, which are subject to an axiom that considers concurrent read and exclusive owner write. A quite general method of recognizing sets of partial orders and graphs is that of graph acceptors as introduced by Thomas [93]. They are known to be exactly as expressive as EMSO logic for arbitrary classes of graphs that have bounded degree. But they lack operational behavior and do not really reflect the dynamic causal nature of a system. We will, however, get to know *asynchronous cellular automata (with types)*, which combine the models of asynchronous automata, graph acceptors, communicating finite-state machines, and many other systems and allow us to treat them in a unifying framework. In particular, asynchronous cellular automata turn out to have the same expressive power as EMSO logic relative to any class of pomsets and dags.

1.5 An Overview of the Book

Chapter 2 recalls some basic notions and results concerning posets, monoids, and formal languages. It moreover presents the well-known halting problem of Turing machines, an undecidable problem that will be used to obtain related undecidability results with respect to communicating automata.

Chapter 3 introduces graphs in general and related notions, presents the corresponding MSO logic to express graph properties, and provides Thomas' fundamental result, which makes use of the famous theorem of Hanf and serves as the basis for upcoming logical characterizations: the expressive equivalence of graph acceptors and EMSO logic.

Chapter 4 recalls the well-known and thoroughly studied model of finite automata over words and their relation with MSO logic and the algebraic notions of recognizability and rationality. Though finite automata are considered to be a purely sequential model, they will, equipped with a communication medium, represent the building blocks of a distributed system.

Chapter 5 lays the foundation of subsequent chapters. It constitutes the basic parameter or architecture of a communicating system in terms of a distributed alphabet and introduces asynchronous cellular automata (with types) as a universal tool unifying finite automata, asynchronous automata, graph acceptors, communicating finite-state machines, and lossy channel systems. Asynchronous cellular automata turn out to be expressively equivalent to EMSO logic relative to dags over distributed alphabets and therefore cover the expressiveness results of all the above-mentioned models. Though, at first sight, asynchronous cellular automata appear as a rather complex and unintuitive model (e.g., compared with finite automata), they have been around since the end of the 80's and are a well-established tool to describe concurrent behavior. In this book, we deal with a particularly simple definition of asynchronous cellular automata to make them accessible to a broad readership. The reader is encouraged to study this model thoroughly; a good comprehension thereof will pay off and enable her to understand more specific concepts more easily (such as shared-memory and channel systems), to design her own automata models and to characterize them logically, and to sharpen the understanding of different phenomena of concurrency and their characteristics.

Chapter 6 presents asynchronous automata, a formal model of shared-memory systems. Naturally, asynchronous automata run over Mazurkiewicz traces, a class of graphs that describe the simultaneous access by several processes to common resources. The literature provides manifold approaches to modeling traces, and every approach has its strengths and weaknesses. In our setting, traces are best defined as graphs, as introduced in Chap. 3. We establish a logical characterization of asynchronous automata in terms of EMSO logic interpreted over traces. Note that the derivation of this equivalence is solely based on our considerations in Chap. 5 and, unlike other methods that have been applied so far, does not rely on further results whose proofs would have gone beyond the scope of this book. Finally, we recall the well-known theorem of Zielonka, which compares asynchronous automata with the notion of recognizability.

From Chap. 7 on, the book concentrates on systems that communicate through reliable or faulty (fifo) channels. In this regard, we first provide the notion of MSCs, followed by the definition of several classes of MSC languages, e.g., generated by high-level MSCs. Recall that a single MSC describes one possible run of the system at hand, whereas a set of MSCs (an MSC language) might be used to characterize the complete system behavior.

Traces are to asynchronous automata as MSCs are to communicating finite-state machines. The definition of communicating finite-state machines and lossy channel systems can be found in Chap. 8, which also deals with

their expressiveness relative to the previously proposed language classes. In particular, again exploiting Chap. 5, we can easily derive from the logical characterization of asynchronous cellular automata a logical characterization of communicating finite-state machines.

Finally, Chap. 9 studies the gap between MSO logic and its existential fragment, exemplified in the framework of communicating finite-state machines, which is also compared to the formalisms of high-level MSCs and recognizability. It will turn out that, as a specification language, the full MSO logic and (compositional) high-level MSCs are too powerful: we identify both MSO and high-level MSC specifications that cannot be implemented in terms of an automaton.

2

Preliminaries

This chapter provides some mathematical background and basic notions concerning binary relations, partial orders, monoids, rational and recognizable languages, and Turing machines.

2.1 Words and Partial Orders

Let Σ be an *alphabet*, i.e., a nonempty finite set of *symbols* or *letters*. The set of (finite) *strings* or *words* over Σ is denoted by Σ^*, the set of nonempty words by Σ^+. The empty string, i.e., the neutral element with respect to word concatenation, is denoted by ε. Note that words will later be seen as system runs and therefore be modeled as a special case of graphs. This is, however, just to treat the objects that represent the behavior of a distributed system in a unified manner and does not affect the notions introduced so far. Thus, we may, throughout the book, consider a word to be a sequence of letters.

Given a set V and binary relations $\mathcal{R}_1, \mathcal{R}_2 \subseteq V \times V$, the *product* $\mathcal{R}_1 \circ \mathcal{R}_2 \subseteq V \times V$ of $\mathcal{R}_1$ and $\mathcal{R}_2$ is the binary relation $\{(u, v) \in V \times V \mid \text{there is } w \in V \text{ such that } (u, w) \in \mathcal{R}_1 \text{ and } (w, v) \in \mathcal{R}_2\}$. Given $\mathcal{R} \subseteq V \times V$, we moreover define $\mathcal{R}^0$ to be $\{(u, u) \mid u \in V\}$ and $\mathcal{R}^{n+1}$ to be $\mathcal{R} \circ \mathcal{R}^n$ for any $n \in \mathbb{N}$ (where $\mathbb{N}$ is the set of natural numbers including 0). Finally, let $\mathcal{R}^*$ denote the infinite union $\bigcup_{n \in \mathbb{N}} \mathcal{R}^n$. Note that, instead of $(u, v) \in \mathcal{R}$, we may also write $u \mathcal{R} v$.

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

- *reflexive* if, for each $u \in V$, $u \leq u$,
- *irreflexive* if, for each $u \in V$, $u \not\leq u$,
- *transitive* if, for any $u, v, w \in V$, $(u \leq v \wedge v \leq w)$ implies $u \leq w$, and
- *antisymmetric* if, for any $u, v \in V$, $(u \leq v \wedge v \leq u)$ implies $u = v$.

As mentioned in the introduction, one single behavior of a distributed system will be described in a compact manner by a partially ordered set.

Definition 2.1 (Partially Ordered Set). *A* (finite) partially ordered set *(also called a* poset*) is a pair* $(V, \leq)$ *such that*

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

Given a poset $(V, \leq)$, we call the relation $\leq$ a *partial order*. A *totally ordered set* is a poset $(V, \leq)$ such that, for any $u, v \in V$, $u \leq v$ or $v \leq u$. Accordingly, we then call the relation $\leq$ a *total order*.

Let $\mathcal{P} = (V, \leq)$ be a poset. By $<$, we denote the binary relation $\leq \setminus \{(u, u) \mid u \in V\}$. Moreover, for $u, v \in V$, let us write $u \lessdot v$ if both $u < v$ and, for any $w \in V$, $u < w \leq v$ implies $w = v$. Then, $(V, \lessdot)$ and $\lessdot$ are called the *Hasse diagram* of $\mathcal{P}$ and, respectively, the *covering relation* of $\leq$. For $u \in V$, we furthermore say that u is *minimal*/*maximal* in $\mathcal{P}$ (we also say minimal/maximal in $(V, <)$) if there is no $v \in V$ such that $v < u / u < v$, respectively. Given an alphabet Σ, a *Σ-labeled poset* is a triple $(V, \leq, \lambda)$ such that $(V, \leq)$ is a poset and λ is a function $V \to \Sigma$, called a *labeling function*.

Throughout this book, we do not distinguish isomorphic structures.

2.2 Monoids and Languages

The objects considered when modeling the behavior of a distributed system are often equipped with a concatenation function, which allows us to combine single behaviors towards more complex ones. Together with a unit element, this gives rise to a monoid.

Definition 2.2 (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 its universe $\mathbb{M}$. A subset of $\mathbb{M}$ is called a *language*. Given languages $L_1, L_2 \subseteq \mathbb{M}$, the *product* of L_1 and L_2 is denoted by $L_1 \cdot L_2$ and defined to be the set $\{t_1 \cdot t_2 \mid t_1 \in L_1 \text{ and } t_2 \in L_2\}$. Furthermore, we set $L^0 := \{\mathbf{1}\}$ and, for $n \in \mathbb{N}$, $L^{n+1} := L \cdot L^n$. 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 book stand for the set of positive natural numbers. A language $L \subseteq \mathbb{M}$ is called *finitely generated* if there is a finite subset Π of $\mathbb{M}$ such that $L \subseteq \langle \Pi \rangle_{\mathbb{M}}$. In that case, we also say that L is *finitely generated by* Π.

Definition 2.3 (Rational Language). *Let* $(\mathbb{M}, \cdot, \mathbf{1})$ *be a monoid. The class* $\mathcal{RAT}_{\mathbb{M}}$ *of* rational *subsets of* $\mathbb{M}$ *is the least set* R *satisfying*

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

Remark 2.4. Any rational language is finitely generated.

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. It may be described by a *rational expression* of $\mathbb{M}$:

Definition 2.5 (Rational Expression). *Let* $(\mathbb{M}, \cdot, \mathbf{1})$ *be a monoid. The set* $\mathrm{REX}_{\mathbb{M}}$ *of* rational expressions *of* $\mathbb{M}$ *is the least set* R *satisfying*

- $\boldsymbol{\emptyset} \in R$,
- $\mathbb{M} \subseteq R$,
- $\alpha_1 \cdot \alpha_2 \in R$ *for any* $\alpha_1, \alpha_2 \in R$,
- $\alpha_1 + \alpha_2 \in R$ *for any* $\alpha_1, \alpha_2 \in R$, *and*
- $\alpha^* \in R$ *for any* $\alpha \in R$.

We call the rational expressions from $\mathbb{M} \cup \{\boldsymbol{\emptyset}\}$ *atomic*, whereas the remaining cases form *compound* expressions. Henceforth, α^+ will stand for $\alpha \cdot \alpha^*$. Moreover, we will often write $\alpha_1\alpha_2$ instead of $\alpha_1 \cdot \alpha_2$. To save brackets, we follow the convention that * binds stronger than the binary operators $\cdot$ and $+$, and $\cdot$ binds stronger than $+$. Let $L(\alpha)$ denote the language that corresponds to the rational expression α of $\mathbb{M}$. It is inductively determined by

- $L(\boldsymbol{\emptyset}) = \emptyset$,
- $L(t) = \{t\}$ 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
- $L(\alpha^*) = L(\alpha)^*$.

Exercise 2.6. Which of the following statements hold for arbitrary monoids $\mathbb{M}$ and arbitrary $\alpha, \alpha_1, \alpha_2 \in \mathrm{REX}_{\mathbb{M}}$? In case that a statement is not true for arbitrary monoids, is there a monoid that makes it true?

(a) $L(\alpha \cdot \boldsymbol{\emptyset}) = \emptyset$,
(b) $L(\alpha) = \emptyset$ implies $\alpha = \boldsymbol{\emptyset}$,
(c) $L(\alpha_1 \cdot \alpha_2) = L(\alpha_2 \cdot \alpha_1)$,
(d) $L((\alpha_1 + \alpha_2)^*) = L(\alpha_1^* + \alpha_2)^*$.

A subset L of $\mathbb{M}$ is called *recognizable* if there exist 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 [79].

Proposition 2.7. *For any monoid* $\mathbb{M}$*, the following are equivalent:*

1. $\mathbb{M}$ *is finitely generated,*
2. $\mathbb{M} \in \mathcal{RAT}_{\mathbb{M}}$*,*
3. $\mathcal{REC}_{\mathbb{M}} \subseteq \mathcal{RAT}_{\mathbb{M}}$*.*

2.3 Turing Machines and the Halting Problem

We recall an important undecidable problem, which will be used to infer several undecidability results in this book: the *halting problem* of a Turing machine.

Definition 2.8. *A* Turing machine *is a structure* $(Q, \Sigma, \Delta, \square, q_0, q_f)$ *where*

- Q *is its nonempty finite set of* states*,*
- Σ *is its (nonempty and finite)* tape alphabet*,*
- $\Delta \subseteq Q \times \Sigma \times \Sigma \times \{L, N, R\} \times Q$ *is its* transition relation*,*
- $\square \in \Sigma \setminus Q$ *is the* blank symbol*, and*
- $q_f \in Q$ *is its* final state*.*

A *configuration* of a Turing machine $\mathcal{M} = (Q, \Sigma, \Delta, \square, q_0, q_f)$ is a word from $(\Sigma \times \{\square\})^*(\Sigma \times Q)(\Sigma \times \{\square\})^*$. Configurations such as $\gamma_0 = (\square, q_0)$, $\gamma_1 = (a, \square)(\square, q_1)$, $\gamma_2 = (a, q_2)(b, \square)$, and $\gamma_3 = (\square, q_3)(a', \square)(b, \square)$ might be written as $\gamma_0 = \begin{pmatrix} \square \\ q_0 \end{pmatrix}$, $\gamma_1 = \begin{pmatrix} a & \square \\ \square & q_1 \end{pmatrix}$, $\gamma_2 = \begin{pmatrix} a & b \\ q_2 & \square \end{pmatrix}$, and $\gamma_3 = \begin{pmatrix} \square & a' & b \\ q_3 & \square & \square \end{pmatrix}$, respectively, or they might be depicted as in Fig. 2.1, which illustrates a possible computation of $\mathcal{M}$. Thus, the first row of a configuration represents the current contents of the *working tape* of $\mathcal{M}$, whereas the second row marks both the position of the letter to be read next and the current state of $\mathcal{M}$ (i.e., in the second row, the symbol $\square$ is just a placeholder and not related to the same symbol that is used in the first row). The set of configurations of $\mathcal{M}$ is denoted by $Conf_{\mathcal{M}}$. Intuitively, a transition $(q, a, a', \theta, q') \in \Delta$ can be read as "if the Turing machine is in state q and reads an a, it may overwrite the a and replace it with an a', move to the left/stand still/move to the right (depending on whether the value of θ is $L/N/R$, respectively), and go into state q'".

The behavior of $\mathcal{M}$ is reflected by a relation $\vdash_{\mathcal{M}} \subseteq Conf_{\mathcal{M}} \times Conf_{\mathcal{M}}$: given $\gamma, \gamma' \in Conf_{\mathcal{M}}$, we write $\gamma \vdash_{\mathcal{M}} \gamma'$ if there are $\underline{w}_1, \underline{w}_2 \in (\Sigma \times \{\square\})^*$, $(q, a, a', \theta, q') \in \Delta$, and $b \in \Sigma$ such that one of the following is true:

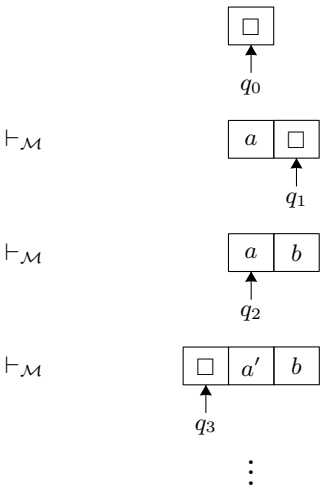


Fig. 2.1. The computation of a Turing machine $\mathcal{M}$

- $\gamma = \underline{w}_1(a,q)\underline{w}_2$, $\gamma' = \underline{w}_1(a',q')\underline{w}_2$, and $\theta = N$,
- $\gamma = \underline{w}_1(a,q)(b,\Box)\underline{w}_2$, $\gamma' = \underline{w}_1(a',\Box)(b,q')\underline{w}_2$, and $\theta = R$,
- $\gamma = \underline{w}_1(a,q)$, $\gamma' = \underline{w}_1(a',\Box)(\Box,q')$, and $\theta = R$,
- $\gamma = \underline{w}_1(b,\Box)(a,q)\underline{w}_2$, $\gamma' = \underline{w}_1(b,q')(a',\Box)\underline{w}_2$, and $\theta = L$,
- $\gamma = (a,q)\underline{w}_2$, $\gamma' = (\Box,q')(a',\Box)\underline{w}_2$, and $\theta = L$.

A sequence of configurations that is in accordance with $\vdash_{\mathcal{M}}$ is illustrated in Fig. 2.1, provided Δ contains at least the transitions $(q_0,\Box,a,R,q_1)$, $(q_1,\Box,b,L,q_2)$, and (q_2,a,a',L,q_3). We say that $\mathcal{M}$ *halts* if there is $\underline{w}_1,\underline{w}_2 \in (\Sigma \times \{\Box\})^*$ and $a \in \Sigma$ such that $(\Box,q_0) \vdash^*_{\mathcal{M}} \underline{w}_1(a,q_f)\underline{w}_2$.

Theorem 2.9. *The following problem is undecidable:*

Input: Turing machine $\mathcal{M}$.
Question: Does $\mathcal{M}$ halt?

The problem described in Theorem 2.9 is (a variant of) the well-known *halting problem*.

2.4 Bibliographic Notes

A comprehensive reference on recognizability and rationality of formal languages over words and posets is [96]. Computability, undecidability, and complexity theory are the subject of [80]. A general introduction to formal languages, Turing machines, and computability is provided by [48].

3

Graphs, Logics, and Graph Acceptors

Directed graphs are the most general structures we consider in this book. Many structures that will also be addressed, such as words and (graphs associated to) posets, can be embedded into graphs or at least have a corresponding one-to-one graph representation.

3.1 Graphs

In the following, let Σ and C be alphabets, which contain the elements the components of a graph are labeled with. Hereby, Σ is the supply of *actions* that a system may execute. The actions will label the nodes of a graph, which we will later refer to as *events*. The elements from C label (color) the edges of a graph and provide a kind of control-flow information.

Definition 3.1 (Graph). *A (directed)* graph *over* (Σ, C) *is a structure* $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda)$ *where*

- V *is its finite set of* nodes,
- *the* $\lhd_\ell \subseteq V \times V$ *are disjoint binary relations on* V*, and*
- $\lambda : V \to \Sigma$ *is the* (node-)labeling function.

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

The set of graphs over (Σ, C) is denoted by $\mathbb{DG}(\Sigma, C)$. Note that we silently assume a graph (actually, its set of nodes) to be nonempty if it seems more convenient, e.g., if we require a mapping on the set of nodes. However, it will always be clear how to extend the setting accordingly to deal with the empty graph.

Let $B \in \mathbb{N}$ be a natural number. For $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$, we say that the degree of $\mathcal{G}$ is *bounded* by B if, for any $u \in V$, $|\{v \in V \mid u \lhd v \text{ or } v \lhd u\}| \leq B$. Given a set $\mathcal{K}$ of graphs over (Σ, C), the *degree* of $\mathcal{K}$ is said to be *bounded* by B if, for any $\mathcal{G} \in \mathcal{K}$, the degree of $\mathcal{G}$ is bounded by B. We say that $\mathcal{K}$ has *bounded degree* if its degree is bounded by some B. By $\mathcal{K}[B]$, we denote the class of graphs $\mathcal{G} \in \mathcal{K}$ such that the degree of $\mathcal{G}$ is bounded by 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 $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda)$ over $(\Sigma \times Q, C)$, i.e, λ is a mapping $V \to \Sigma \times Q$. Note that λ can be seen as a pair (λ', ρ) of mappings $V \to \Sigma$ and $V \to Q$, respectively. Given a class $\mathcal{K}$ of graphs over (Σ, C) and a (possibly empty) finite set Q, we define $\langle \mathcal{K}, Q \rangle$ to be $\mathcal{K}$ if Q is empty and, otherwise, to be the set of Q-extended graphs $(V, \{\lhd_\ell\}_{\ell \in C}, (\lambda, \rho))$ over (Σ, C) such that $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathcal{K}$. If Q is nonempty and we are given $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ and a mapping $\rho : V \to Q$, then we write $(\mathcal{G}, \rho)$ to denote the extended graph $(V, \{\lhd_\ell\}_{\ell \in C}, (\lambda, \rho)) \in \langle \mathbb{DG}(\Sigma, C), Q \rangle$.

3.2 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 MSO logic over words, traces, and message sequence charts. Fragments of MSO logic will provide logical characterizations of respective automata models. For a comprehensive overview of MSO logics, see [39].

Throughout the book, we will use 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 3.2 (Monadic Second-Order Logic). *The set* $\mathrm{MSO}(\Sigma, C)$ *of* monadic second-order formulas *over* (Σ, C) *(or* $\mathrm{MSO}(\Sigma, C)$*-formulas) is built up from the atomic formulas*

- $\lambda(x) = a$ *(with* $x \in \mathrm{Var}$ *and* $a \in \Sigma$*),*
- $x \lhd_\ell y$ *(with* $x, y \in \mathrm{Var}$ *and* $\ell \in C$*),*
- $x \in X$ *(with* $x \in \mathrm{Var}$ *and* $X \in \mathrm{VAR}$*), and*
- $x = y$ *(with* $x, y \in \mathrm{Var}$*),*

and, furthermore, allow the boolean connectives $\neg$, $\vee$, $\wedge$, $\to$, *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) quantifiers, respectively. More precisely, if* φ *and* ψ *are formulas from* $\mathrm{MSO}(\Sigma, C)$*, then so are* $\neg\varphi$*,*

$\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \to \psi$, $\varphi \leftrightarrow \psi$, $\exists x\varphi$, $\forall x\varphi$, $\exists X\varphi$, *and* $\forall X\varphi$ *(where* $x \in \mathrm{Var}$ *and* $X \in \mathrm{VAR}$*).*

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

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

- $\mathcal{G} \models_{\mathcal{I}} \neg\varphi$ if *not* $\mathcal{G} \models_{\mathcal{I}} \varphi$,
- $\mathcal{G} \models_{\mathcal{I}} \varphi \vee \psi$ if $\mathcal{G} \models_{\mathcal{I}} \varphi$ *or* $\mathcal{G} \models_{\mathcal{I}} \psi$,
- $\mathcal{G} \models_{\mathcal{I}} \varphi \wedge \psi$ if $\mathcal{G} \models_{\mathcal{I}} \varphi$ *and* $\mathcal{G} \models_{\mathcal{I}} \psi$,
- $\mathcal{G} \models_{\mathcal{I}} \varphi \to \psi$ if *not* $\mathcal{G} \models_{\mathcal{I}} \varphi$ *or* $\mathcal{G} \models_{\mathcal{I}} \psi$,
- $\mathcal{G} \models_{\mathcal{I}} \varphi \leftrightarrow \psi$ if $\mathcal{G} \models_{\mathcal{I}} \varphi$ *iff* $\mathcal{G} \models_{\mathcal{I}} \psi$,

- $\mathcal{G} \models_{\mathcal{I}} \exists x\varphi$ if there is $u \in V$ such that $\mathcal{G} \models_{\mathcal{I}[x/u]} \varphi$,
- $\mathcal{G} \models_{\mathcal{I}} \forall x\varphi$ if, for any $u \in V$, $\mathcal{G} \models_{\mathcal{I}[x/u]} \varphi$,
- $\mathcal{G} \models_{\mathcal{I}} \exists X\varphi$ if there is $V' \subseteq V$ such that $\mathcal{G} \models_{\mathcal{I}[X/V']} \varphi$, and
- $\mathcal{G} \models_{\mathcal{I}} \forall X\varphi$ if, for any $V' \subseteq V$, $\mathcal{G} \models_{\mathcal{I}[X/V']} \varphi$.

Here, $\mathcal{I}[x/u]$ is the mapping that coincides with $\mathcal{I}$ except in x, which is mapped by $\mathcal{I}[x/u]$ to u. Accordingly, $\mathcal{I}[X/V']$ maps X to V' and, otherwise, coincides with $\mathcal{I}$.

To save brackets, we stipulate that, as usual, the unary operators bind stronger than the binary ones. Moreover, $\wedge$ binds stronger than $\vee$, which, in turn, binds stronger than the combinators $\to$ and $\leftrightarrow$.

We introduce some abbreviations and let $x \lhd y$ stand for the $\mathrm{MSO}(\Sigma, C)$-formula $\bigvee_{\ell \in C} x \lhd_\ell y$. Moreover, for some $\ell \in C$, $x \leq_\ell y$ and $x \leq y$ will abbreviate

$$\forall X\,(x \in X \wedge \forall z \forall z'\,((z \in X \wedge z \lhd_\ell z') \to z' \in X) \to y \in X)$$

and, respectively,

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

whereas $x <_\ell y$ and $x < y$ are shorthands for the formulas $x \leq_\ell y \wedge \neg(x = y)$ and $x \leq y \wedge \neg(x = y)$, respectively. Finally, given $\Sigma' \subseteq \Sigma$, $\lambda(x) \in \Sigma'$ shall abbreviate $\bigvee_{a \in \Sigma} \lambda(x) = a$. Note that $x \leq y$ could have been defined equivalently by

$$\exists X\,(x \in X \wedge y \in X \wedge \forall z(z \in X \to z = y \vee \exists z'(z' \in X \wedge z \lhd z')))$$

as well, which is an *existential* MSO formula as explained below.

If we consider *sentences*, i.e., formulas without free variables, we accordingly replace $\models_{\mathcal{I}}$ with $\models$, as satisfaction of a sentence is independent of a given interpretation function.

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 (Σ, 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 $\varphi(X_1, \ldots, X_n, \overline{Y})$ is contained in $\mathrm{FO}(\Sigma, C)$. Throughout the book, we let $\mathrm{EMSO}(\Sigma, C)$ denote the class of existential $\mathrm{MSO}(\Sigma, C)$-formulas.

Concerning second-order variables, we would generally like to distinguish formulas by their quantifier-alternation depth. Namely, for $k \in \mathbb{N}_{\geq 1}$, $\Sigma_k(\Sigma, C)$ 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 $\varphi(\overline{X_1}, \ldots, \overline{X_k}, \overline{Y})$ (the $\overline{X_i}$ 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

- $\lambda(x) = a$,
- $x \leq y$,
- $x \in X$, and
- $x = y$

yields in the canonical manner the logics $\mathrm{MSO}(\Sigma, C)[\leq]$, $\mathrm{EMSO}(\Sigma, C)[\leq]$, $\Sigma_k(\Sigma, C)[\leq]$, and $\mathrm{FO}(\Sigma, C)[\leq]$, respectively. The semantics of $x \leq y$ with respect to a given graph $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ and a corresponding interpretation function $\mathcal{I}$ is determined by $\mathcal{G} \models_{\mathcal{I}} x \leq y$ if $\mathcal{I}(x) \lhd^* \mathcal{I}(y)$. Other logics arise in a similar manner. Restricting to the 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.

We introduce another abbreviation: for set variables $X_1, \ldots, X_n$ $(n \geq 1)$, the first-order formula

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

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

Let $\mathcal{K}$ be a set of graphs over (Σ, 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

$\mathcal{G} \in \mathcal{K}$ with $\mathcal{G} \models \varphi$. A formula $\varphi(x_1, \ldots, x_m, X_1, \ldots, X_n) \in \mathrm{MSO}(\Sigma, C)$ (potentially with free variables) actually defines a language of graphs whose labelings are enriched by tuples from $\{0,1\}^{m+n}$. So let $\langle \mathcal{K}, \{0,1\}^{m,n} \rangle$ denote $\mathcal{K}$ if $m = n = 0$ and, otherwise, the set of $\{0,1\}^{m+n}$-extended graphs $(V, \{\lhd_\ell\}_{\ell \in C}, (\lambda, \rho)) \in \mathbb{DG}(\Sigma \times \{0,1\}^{m+n}, C)$ such that, for any $i \in \{1, \ldots, m\}$, there is exactly one $u \in V$ with $\rho(u)[i] = 1$ (where $\rho(u)[i]$ yields the i-th component of $\rho(u)$). Canonically, we will abbreviate $\langle \mathcal{K}, \{0,1\}^{0,n} \rangle$ by $\langle \mathcal{K}, \{0,1\}^{n} \rangle$. We may synonymously write both $L_{\mathcal{K}}(\varphi)$ and $L_{\langle \mathcal{K}, \{0,1\}^{m,n} \rangle}(\varphi)$ to denote the language of φ relative to $\mathcal{K}$, which is then a subset of $\langle \mathcal{K}, \{0,1\}^{m,n} \rangle$. More precisely, an extended graph $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, (\lambda, \rho)) \in \langle \mathcal{K}, \{0,1\}^{m,n} \rangle$ *satisfies* φ if we have $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \models_{\mathcal{I}_\mathcal{G}} \varphi$ where $\mathcal{I}_\mathcal{G}(x_i) = u$ if $\rho(u)[i] = 1$ and $u \in \mathcal{I}_\mathcal{G}(X_i)$ if $\rho(u)[m+i] = 1$.

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, with respect to the alternative predicate symbol $\leq$, we accordingly 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. In general, the classes of $\Sigma_k(\Sigma, C)_{\mathbb{DG}(\Sigma, C)}$-definable languages form an infinite hierarchy [65, 66]. 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. In that case, we speak of graphs over $(\Sigma, -)$ or over $(-, C)$ and respectively write, for example, $\mathbb{DG}(\Sigma, -)$ and $\mathcal{FO}(-, C)[\leq]_\mathcal{K}$. If C is a singleton, we may even speak of a graph over Σ and write, for example, $\mathbb{DG}(\Sigma)$ instead of $\mathbb{DG}(\Sigma, -)$. 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, we will consider $\mathrm{MSO}_0(\Sigma, C)$, another slightly modified logic, which, in contrast to the logic $\mathrm{MSO}(\Sigma, C)[\leq]$, has in general the same expressive power as $\mathrm{MSO}(\Sigma, C)$. Its atomic entities are

- $\lambda(X) \subseteq \{a\}$,
- $X \lhd_\ell Y$,
- $X \subseteq Y$, and
- $\mathit{Sing}(X)$,

where, as usual, $a \in \Sigma$, $\ell \in C$, and $X, Y \in \mathrm{VAR}$. 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 $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ and an interpretation $\mathcal{I}$,

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

Following our convention, $\mathcal{MSO}_0(\Sigma, C)_{\mathcal{K}}$ with $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ denotes 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 3.3.

$$\mathcal{MSO}_{\mathbb{DG}} \;=\; (\mathcal{MSO}_0)_{\mathbb{DG}}$$

Exercise 3.4.

(a) Prove Lemma 3.3.
(b) Verify that, for any class $\mathcal{K} \subseteq \mathbb{DG}$, $\mathcal{MSO}_{\mathcal{K}} = (\mathcal{MSO}_0)_{\mathcal{K}}$.

Let us discuss further relations between the language classes proposed so far.

Lemma 3.5. *If $L \subseteq \mathbb{DG}$ is $\mathrm{MSO}_{\mathbb{DG}}$-definable, then it is $(\Sigma_k)_{\mathbb{DG}}$-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 relative to $\mathbb{DG}$. 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. □

Lemma 3.6.

$$\mathcal{MSO}[\leq]_{\mathbb{DG}} \;\subseteq\; \mathcal{MSO}_{\mathbb{DG}}$$

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 equivalent MSO-formula. □

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

Trivially, the sets of FO-, EMSO-, and MSO-definable languages form a hierarchy. The inverse inclusions do not hold in general. However, we will identify classes of graphs for which EMSO logic is already as expressive as MSO logic.

Lemma 3.7.

(a) $\mathcal{FO}_{\mathbb{DG}} \subseteq \mathcal{EMSO}_{\mathbb{DG}} \subseteq \mathcal{MSO}_{\mathbb{DG}}$,
(b) $\mathcal{FO}[\leq]_{\mathbb{DG}} \subseteq \mathcal{EMSO}[\leq]_{\mathbb{DG}} \subseteq \mathcal{MSO}[\leq]_{\mathbb{DG}}$.

3.3 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 $\mathcal{H}$ hereby specifies the local neighborhood around a distinguished center γ where the size of the neighborhood is constituted by a natural number $R \in \mathbb{N}$, the *radius* of $\mathcal{H}$, which restricts the distance of any node of $\mathcal{H}$ to γ.

Let us make this idea more precise and let R be a natural number. Given a graph $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ and nodes $u, v \in V$, the *distance* $d_{\mathcal{G}}(v, u)$ from v to u in $\mathcal{G}$ is ∞ if it holds $(u, v) \notin (\lhd \cup \lhd^{-1})^*$ and, otherwise, the minimal natural number k such that there is a sequence of elements $u_0, \ldots, u_k \in V$ with $u_0 = u$, $u_k = v$, and $u_i \lhd u_{i+1}$ or $u_{i+1} \lhd u_i$ for each $i \in \{0, \ldots, k-1\}$. Sometimes, if it is clear from the context, we omit the subscript $\mathcal{G}$, just writing $d(v, u)$.

Definition 3.8 (R-Sphere). *An* R-sphere *over* (Σ, C) *is a structure* $\mathcal{H} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma)$ *where* $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda)$ *is a graph over* (Σ, C) *and* $\gamma \in V$ *is a designated* sphere center *such that, for any* $u \in V$, $d_{\mathcal{H}}(u, \gamma) \leq R$ *(in slight abuse of notation, the distance from one node to another can be given with respect to a sphere as well).*

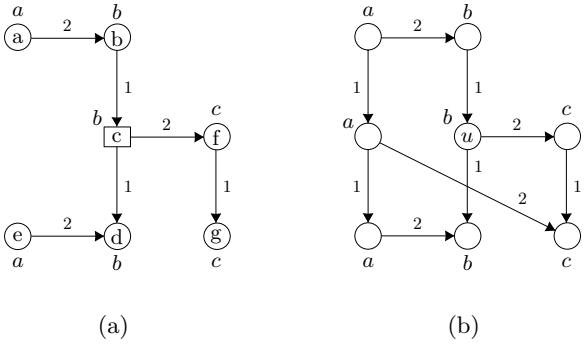


Fig. 3.1. A 2-sphere over $(\{a, b, c\}, \{1, 2\})$

Let $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$. For $u \in V$, we set the *R-sphere of $\mathcal{G}$ around u*, denoted by $R\text{-Sph}(\mathcal{G}, u)$, to be $(V', \{\lhd'_\ell\}_{\ell \in C}, \lambda', u)$ where $V' =$

$\{v \in V \mid d_{\mathcal{G}}(v,u) \leq R\}$, $\lhd'_\ell = \lhd_\ell \cap (V' \times V')$ for each $\ell \in C$, and λ' is the restriction of λ to V'. Given an R-sphere $\mathcal{H}$ over (Σ, C), we define $|\mathcal{G}|_{\mathcal{H}}$ to be $|\{u \in V \mid R\text{-Sph}(\mathcal{G}, u) \cong \mathcal{H}\}|$, i.e., the number of occurrences of $\mathcal{H}$ in $\mathcal{G}$. (Here and henceforth, $\cong$ stands for the isomorphism relation.)

A 2-sphere over $(\{a,b,c\},\{1,2\})$ is shown in Fig. 3.1a where the sphere center is depicted as a rectangle. It precisely deals with the 2-sphere of the graph from Fig. 3.1b around u.

Given $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$, we set $R\text{-Sph}(\mathcal{K})$, the set of R-spheres that arise from $\mathcal{K}$, to be $\{R\text{-Sph}(\mathcal{G}, u) \mid \mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathcal{K}, u \in V\}$.

In the context of spheres, the *rank* of a first-order formula will play a crucial role. The rank $rank(\varphi)$ of $\varphi \in \text{FO}(\Sigma, C)$ is the maximal number of nested first-order quantifiers. Formally, it is defined inductively via

- $rank(\varphi) = 0$ if φ is atomic,
- $rank(\neg\varphi) = rank(\varphi)$,
- $rank(\varphi\ \theta\ \psi) = \max\{rank(\varphi), rank(\psi)\}$ for $\theta \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$,
- $rank(\exists x\varphi) = rank(\varphi) + 1$, and
- $rank(\forall x\varphi) = rank(\varphi) + 1$.

For $k \in \mathbb{N}$, in the following, let $\text{FO}_k(\Sigma, C)$ comprise the formulas $\varphi \in \text{FO}(\Sigma, C)$ such that $rank(\varphi) \leq k$.

The rank of a first-order sentence φ allows us to compute a radius R such that the satisfaction of φ in a graph $\mathcal{G}$ is determined solely by the mapping that reveals, for any R-sphere $\mathcal{H}$, how often $\mathcal{H}$ occurs in $\mathcal{G}$. In other words, when, for any R-sphere $\mathcal{H}$, we are told how often $\mathcal{H}$ occurs in $\mathcal{G}$ as a pattern, we can say whether $\mathcal{G} \models \varphi$ or $\mathcal{G} \not\models \varphi$. Basically, this what we know as Hanf's theorem. Before we state this central theorem formally, we need to introduce two equivalence relations. The first is parameterized by a natural number k and relates graphs that cannot be distinguished by formulas of rank at most k. The second counts the number of spheres in a graph up to some threshold.

Definition 3.9 (FO-k-Equivalence). *Let $k \in \mathbb{N}$. Given graphs $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{DG}(\Sigma, C)$, we write $\mathcal{G}_1 \equiv_k \mathcal{G}_2$ if, for any sentence $\varphi \in \text{FO}_k(\Sigma, C)$, we have $\mathcal{G}_1 \models \varphi$ iff $\mathcal{G}_2 \models \varphi$.*

Lemma 3.10. *For any $k \in \mathbb{N}$, $\equiv_k$ has finite index, i.e., $\{[\mathcal{G}]_{\equiv_k} \mid \mathcal{G} \in \mathbb{DG}(\Sigma, C)\}$ is a finite set.*

Proof. We show by induction on k that, for any natural number $m \in \mathbb{N}$, it holds that $\{L_{\langle \mathbb{DG}(\Sigma,C), \{0,1\}^{m,0}\rangle}(\varphi) \mid \varphi(x_1, \ldots, x_m) \in \text{FO}_k(\Sigma, C)\}$ is a finite set, which implies the lemma. Of course, there are only finitely many atomic first-order formulas and finitely many of their boolean combinations up to logical equivalence. Moreover, any formula from $\text{FO}_{k+1}(\Sigma, C)$ with at most m free individual variables can be written as the boolean combination of formulas of the form $\exists x_{m+1}\psi$ with $\psi(x_1, \ldots, x_m, x_{m+1}) \in \text{FO}_k(\Sigma, C)$. By the induction hypothesis, we have that $\{L_{\langle \mathbb{DG}(\Sigma,C), \{0,1\}^{m+1,0}\rangle}(\psi') \mid \psi'(x_1, \ldots, x_m, x_{m+1}) \in \text{FO}_k(\Sigma, C)\}$ is finite and, thus, so is $\{L_{\langle \mathbb{DG}(\Sigma,C), \{0,1\}^{m,0}\rangle}(\varphi) \mid \varphi(x_1, \ldots, x_m) \in \text{FO}_{k+1}(\Sigma, C)\}$. □

The union of relations $\equiv_k$ is complete for characterizing FO-definability:

Lemma 3.11. *Let $L \subseteq \mathbb{DG}(\Sigma, C)$. We have $L \in \mathcal{FO}(\Sigma, C)_{\mathbb{DG}}$ iff there is some $k \in \mathbb{N}$ such that L is the union of $\equiv_k$-equivalence classes.*

Exercise 3.12. Prove Lemma 3.11.

Definition 3.13 (Threshold Equivalence). *Let $R, t \in \mathbb{N}$. Given graphs $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{DG}(\Sigma, C)$, we write $\mathcal{G}_1 \leftrightarrows_{R,t} \mathcal{G}_2$ if, for any R-sphere $\mathcal{H}$ over (Σ, C), either*

- *$|\mathcal{G}_1|_{\mathcal{H}} = |\mathcal{G}_2|_{\mathcal{H}}$ or*
- *both $|\mathcal{G}_1|_{\mathcal{H}} \geq t$ and $|\mathcal{G}_2|_{\mathcal{H}} \geq t$.*

In other words, $\leftrightarrows_{R,t}$ distinguishes graphs on the basis of the number of R-spheres up to some threshold t.

Exercise 3.14. Let $B \in \mathbb{N}$. Show that, for any $R, t \in \mathbb{N}$, $\leftrightarrows_{R,t}$ has finite index if we consider only graphs whose degree is bounded by B.

Theorem 3.15 (Hanf [41]). *Let $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ be a class of bounded degree. For any $k \in \mathbb{N}$, one can compute $R, t \in \mathbb{N}$ such that, for any $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{K}$,*

$$\mathcal{G}_1 \leftrightarrows_{R,t} \mathcal{G}_2 \quad \textit{implies} \quad \mathcal{G}_1 \equiv_k \mathcal{G}_2$$

A proof of Theorem 3.15 can be found in [94]. There, given $k \in \mathbb{N}$, R is determined to be 3^k and t is determined to be $k \cdot c$ where c is the maximal size of a 3^k-sphere.

Thus, threshold equivalence may be used to show that a property is not expressible in first-order logic. More precisely, given $L, \mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ where $\mathcal{K}$ has bounded degree, $L \notin \mathcal{FO}(\Sigma, C)_{\mathcal{K}}$ if, for any $R, t \in \mathbb{N}$, we can find $\mathcal{G}_1 \in L$ and $\mathcal{G}_2 \in \mathcal{K} \setminus L$ such that $\mathcal{G}_1 \leftrightarrows_{R,t} \mathcal{G}_2$.

Basically, Hanf's Theorem states that any first-order sentence can be rephrased as a boolean combination of conditions "R-sphere $\mathcal{H}$ occurs at least $n \in \mathbb{N}$ times". We would like to put this idea into a formal framework and first let $\mathcal{O}$ be an arbitrary set (of *objects*). A pair $(o, n) \in \mathcal{O} \times \mathbb{N}$ shall henceforth represent the statement "o occurs at least $n \in \mathbb{N}$ times" and will therefore be written more suggestively as $o \geq n$. In the following, those conditions will be put together towards boolean combinations.

In general, given a set $\mathcal{O}$ and a natural number $t \in \mathbb{N}$, the set $\mathit{Cond}(\mathcal{O}, t)$ of *occurrence formulas* or *occurrence conditions* over $\mathcal{O}$ and t is inductively defined via

- $\mathcal{O} \times \{0, \ldots, t\} \subseteq \mathit{Cond}(\mathcal{O}, t)$,
- $\neg\alpha \in \mathit{Cond}(\mathcal{O}, t)$ if $\alpha \in \mathit{Cond}(\mathcal{O}, t)$,
- $\alpha_1 \vee \alpha_2 \in \mathit{Cond}(\mathcal{O}, t)$ if $\alpha_1, \alpha_2 \in \mathit{Cond}(\mathcal{O}, t)$, and
- $\alpha_1 \wedge \alpha_2 \in \mathit{Cond}(\mathcal{O}, t)$ if $\alpha_1, \alpha_2 \in \mathit{Cond}(\mathcal{O}, t)$.

An occurrence formula $\alpha \in \mathit{Cond}(\mathcal{O}, t)$ is canonically interpreted over a mapping $m : \mathcal{O} \to \mathbb{N}$. Precisely,

- $m \models o \geq n$ if $m(o) \geq n$,
- $m \models \neg\alpha$ if *not* $m \models \alpha$,
- $m \models \alpha \vee \beta$ if $m \models \alpha$ *or* $m \models \beta$, and
- $m \models \alpha \wedge \beta$ if $m \models \alpha$ *and* $m \models \beta$.

Let $\mathcal{S}$ be a set of R-spheres over (Σ, C). An occurrence formula from $\mathit{Cond}(\Sigma, t)$ is *satisfied* by the graph $\mathcal{G} \in \mathbb{DG}(\Sigma, C)$, written $\mathcal{G} \models \alpha$, if α is satisfied by the mapping that assigns $|\mathcal{G}|_a$ to any $a \in \Sigma$. Accordingly, an occurrence formula $\alpha \in \mathit{Cond}(\mathcal{S}, t)$ is satisfied by the graph $\mathcal{G}$ if it is satisfied by the mapping that assigns to each R-sphere $\mathcal{H} \in \mathcal{S}$ the natural number $|\mathcal{G}|_{\mathcal{H}}$. Relative to a set of graphs $\mathcal{K}$, the latter notion gives rise to the class of *locally threshold testable* graph languages:

Definition 3.16 (Locally Threshold Testable). *A set $L \subseteq \mathbb{DG}(\Sigma, C)$ is called* locally threshold testable *relative to a set $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ if there exist $R, t \in \mathbb{N}$ and $\alpha \in \mathit{Cond}(R\text{-Sph}(\mathbb{DG}(\Sigma, C)), t)$ such that $L = \{\mathcal{G} \in \mathcal{K} \mid \mathcal{G} \models \alpha\}$. The corresponding language class is denoted by $\mathcal{LTT}(\Sigma, C)_{\mathcal{K}}$.*

Theorem 3.17. *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 $L \in \mathcal{LTT}(\Sigma, C)_{\mathcal{K}}$.*

Proof. Suppose $L \in \mathcal{FO}(\Sigma, C)_{\mathcal{K}}$ for some $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ whose degree is bounded by, say, $B \in \mathbb{N}$. There exist $k \in \mathbb{N}$ and a sentence $\varphi \in \mathrm{FO}_k(\Sigma, C)$ such that $L = L_{\mathcal{K}}(\varphi)$. According to Lemmata 3.10 and 3.11, L is the finite union $L_1 \cup \ldots \cup L_l$ of $\equiv_k$-equivalence classes. In turn, there are $R, t \in \mathbb{N}$ such that any set L_i, $i = 1, \ldots, l$, is the finite union $L_{i_1} \cup \ldots \cup L_{i_{l_i}}$ of $\leftrightarrows_{R,t}$-equivalence classes (cf. Theorem 3.15). Any $\leftrightarrows_{R,t}$-equivalence class will be captured by an occurrence condition from $\mathit{Cond}(R\text{-Sph}(\mathbb{DG}[B]), t)$. So let $i \in \{1, \ldots, l\}$ and $j \in \{1, \ldots, l_i\}$ and suppose $\mathcal{G} \in L_{i_j}$. Then, $\alpha_{i_j} \in \mathit{Cond}(R\text{-Sph}(\mathbb{DG}[B]), t)$ capturing L_{i_j} shall be given as follows:

$$\alpha_{i_j} \;=\; \bigwedge_{\substack{\mathcal{H} \in R\text{-Sph}(\mathbb{DG}[B]) \\ |\mathcal{G}|_{\mathcal{H}} \geq t}} \mathcal{H} \geq t \quad \wedge \bigwedge_{\substack{\mathcal{H} \in R\text{-Sph}(\mathbb{DG}[B]) \\ |\mathcal{G}|_{\mathcal{H}} = n < t}} (\mathcal{H} \geq n \;\wedge\; \neg\,(\mathcal{H} \geq n + 1))$$

Note that α_{i_j} does not depend on the choice of $\mathcal{G}$ as a representative of L_{i_j}, i.e., we may choose an arbitrary $\mathcal{G} \in L_{i_j}$. Finally, set α to be

$$\alpha \;=\; \bigvee_{\substack{i \in \{1,\ldots,l\} \\ j \in \{1,\ldots,l_i\}}} \alpha_{i_j}$$

Clearly, we have $L = \{\mathcal{G} \in \mathcal{K} \mid \mathcal{G} \models \alpha\}$, which concludes the proof. □

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 $n \geq 1$. Given an extended graph $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DG}(\Sigma \times \{0,1\}^n, C)$ over (Σ, C), the *projection* of $\mathcal{G}$ is defined to be $h(\mathcal{G}) := (V, \{\lhd_\ell\}_{\ell \in C}, \lambda') \in \mathbb{DG}(\Sigma, C)$ where, for any $u \in V$, we have $\lambda'(u) = a$ if $\lambda(u) = (a, (b_1, \ldots, b_n))$ for some $b_1, \ldots, b_n \in \{0,1\}$. The projection function h is canonically extended towards graph languages $L \subseteq \mathbb{DG}(\Sigma \times \{0,1\}^n, C)$, i.e., $h(L) := \{h(\mathcal{G}) \mid \mathcal{G} \in L\}$. Recall that a formula $\varphi(X_1, \ldots, X_n) \in \mathrm{MSO}(\Sigma, C)$ can be interpreted with respect to $\mathcal{G}$ by inferring from the additional labelings an interpretation function $\mathcal{I}_\mathcal{G}$, which assigns to variable X_i the set of all those nodes whose labeling in the i-th component equals 1 so that we may write $\mathcal{G} \models \varphi(X_1, \ldots, X_n)$ if $h(\mathcal{G}) \models_{\mathcal{I}_\mathcal{G}} \varphi(X_1, \ldots, X_n)$.

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

Proposition 3.18 ([92]). *For any class $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$ of bounded degree, $\mathcal{EMSO}(\Sigma, C)_\mathcal{K} = \bigcup_{n \geq 1} \{h(L) \mid L \in \mathcal{LTT}(\Sigma \times \{0,1\}^n, C)_{\langle \mathcal{K}, \{0,1\}^n \rangle}\}$.*

Proof. Recall that a language $L \subseteq \mathbb{DG}(\Sigma, C)$ is $\mathrm{EMSO}(\Sigma, C)_\mathcal{K}$-definable iff it is the language of some $\mathrm{EMSO}(\Sigma, C)$-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 $\mathrm{FO}(\Sigma \times \{0,1\}^n, C)$ (replace any $x \in X_i$ with $\bigvee_{b_i = 1} \lambda(x) = (a, (b_1, \ldots, b_n))$, $L_{\langle \mathcal{K}, \{0,1\}^n \rangle}(\varphi) \in \mathcal{FO}(\Sigma \times \{0,1\}^n, C)_{\langle \mathcal{K}, \{0,1\}^n \rangle}$ is, according to Theorem 3.17, a locally threshold testable set, whose projection $h(L_{\langle \mathcal{K}, \{0,1\}^n \rangle}(\varphi))$ coincides with L. □

3.4 Graph Acceptors

Graph acceptors [91, 93] are a generalization of finite automata to graphs. They are known to be expressively equivalent to EMSO logic with respect to 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 3.19 (Graph Acceptor [91, 93]).
A graph acceptor *over (Σ, C) is a structure $\mathcal{B} = (Q, R, \mathcal{S}, Occ)$ where*

- *Q is its nonempty finite set of* states*,*
- *$R \in \mathbb{N}$ is the* radius*,*
- *$\mathcal{S}$ is a nonempty finite set of R-spheres over $(\Sigma \times Q, C)$, and*
- *$Occ \in Cond(\mathcal{S}, t)$ for some $t \in \mathbb{N}$.*

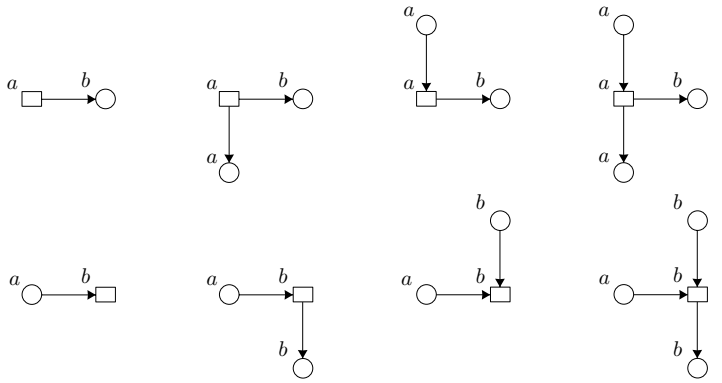


Fig. 3.2. A graph acceptor over $\{a, b\}$

The set of spheres $\mathcal{S}$ of a graph acceptor over $(\{a, b\}, -)$ with $R = 1$ is depicted in Fig. 3.2 (here, the set of states is supposed to be a singleton).

So let $\mathcal{B} = (Q, R, \mathcal{S}, Occ)$ be a graph acceptor over (Σ, C). A *run* of $\mathcal{B}$ on a graph $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ is a mapping $\rho : V \to Q$ such that, for each $u \in V$, the R-sphere of $(V, \{\lhd_\ell\}_{\ell \in C}, (\lambda, \rho))$ around u is isomorphic to some $\mathcal{H} \in \mathcal{S}$. We call ρ *accepting* if $(\mathcal{G}, \rho) \models Occ$. 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 $\mathcal{G} \in \mathcal{K}$ 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 $(\Sigma, C)\}$.

A run of the graph acceptor from Fig. 3.2 is depicted in Fig. 3.3 (again, the only state is omitted).

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 $k\text{-}\mathcal{GA}(\Sigma, C)_\mathcal{K}$ or $k\text{-}\mathcal{GA}_\mathcal{K}$ the class $\{L \subseteq \mathcal{K} \mid L = L_\mathcal{K}(\mathcal{B})$ for some graph acceptor $\mathcal{B} = (Q, R, \mathcal{S}, Occ)$ over (Σ, C) with $R = k\}$.

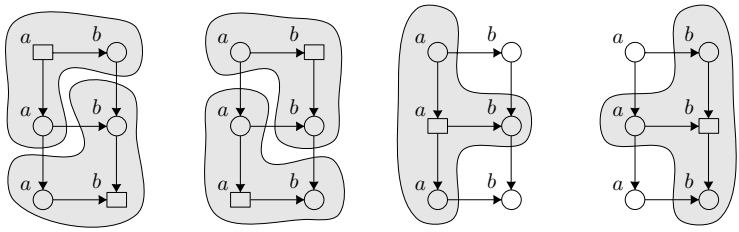


Fig. 3.3. The run of a graph acceptor

Lemma 3.20. *For any $k \in \mathbb{N}_{\geq 1}$ and any class $\mathcal{K} \subseteq \mathbb{DG}$ of bounded degree,*

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

Exercise 3.21. Prove Lemma 3.20.

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

Lemma 3.22. *In general,*

$$1\text{-}\mathcal{GA}_{\mathcal{K}} \subsetneq \mathcal{GA}_{\mathcal{K}}$$

The proof of Lemma 3.22 is deferred to Sect. 3.6.

Given $\mathcal{K} \subseteq \mathbb{DG}(\Sigma, C)$, $\mathcal{GA}(\Sigma, C)^-_{\mathcal{K}}$ shall denote the class of sets $L \subseteq \mathcal{K}$ such that there is a graph acceptor $\mathcal{B} = (Q, R, \mathcal{S}, Occ)$ over (Σ, C) with both $Occ = \mathcal{H} \geq 0$ for some $\mathcal{H} \in \mathcal{S}$ and $(L = L_{\mathcal{K}}(\mathcal{B})$ or $L = L_{\mathcal{K}}(\mathcal{B}) \setminus \{(\emptyset, \{\emptyset\}_{\ell \in C}, \emptyset)\})$. The latter is because a graph acceptor "without occurrence constraints" has no means to reject the empty graph, unless it is not contained in $\mathcal{K}$. The class $k\text{-}\mathcal{GA}(\Sigma, C)^-_{\mathcal{K}}$ with $k \in \mathbb{N}$ is defined accordingly.

Lemma 3.23. *In general, $1\text{-}\mathcal{GA}_{\mathcal{K}} \setminus \mathcal{GA}^-_{\mathcal{K}}$ is not empty.*

Proof. Consider graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ over $(\{a, b\}, -)$ where $\mathcal{G}_1$ and $\mathcal{G}_2$ each consist of one node, which is labeled with a and b, respectively. Then, $\{\mathcal{G}_1, \mathcal{G}_2\}$ 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 $\mathcal{G}_1$ and $\mathcal{G}_2$ also admits an accepting run on the union of $\mathcal{G}_1$ and $\mathcal{G}_2$ (which is obtained in the obvious manner). On the other hand, $\{\mathcal{G}_1, \mathcal{G}_2\}$ is the language of some graph acceptor with radius 1 (even radius 0) where an occurrence constraint ensures that the union of $\mathcal{G}_1$ and $\mathcal{G}_2$ is excluded from the recognized language. □

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 $\mathcal{K}$.

Let us now compare EMSO logic to the formalism of graph acceptors. The following theorem provides the basis for characterizations of further automata models on graphs.

Theorem 3.24 (Thomas [92, 93]). *For any class $\mathcal{K} \subseteq \mathbb{DG}$ of bounded degree,*

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

Proof. The equivalence directly follows from Proposition 3.18. 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. □

3.5 Directed Acyclic Graphs

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. This view motivates the following definition:

Definition 3.25 (Directed Acyclic Graph). *A* directed acyclic graph *(dag) over* (Σ, C) *is a graph* $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda) \in \mathbb{DG}(\Sigma, C)$ *such that* $\lhd$ *is irreflexive and* $\leq$ *is a partial order.*

The set of all those dags is denoted by $\mathbb{DAG}(\Sigma, C)$. A useful subclass of $\mathbb{DAG}(\Sigma, C)$, denoted by $\mathbb{DAG}_H(\Sigma, C)$, is the set of graphs $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda) \in \mathbb{DAG}(\Sigma, C)$ such that $\lhd = \lessdot$, i.e., $(V, \lhd)$ is the Hasse diagram of some partially ordered set. Usually, this will be required if the set of edge labelings is a singleton. Recall that, throughout the book, the nodes of a graph are also called *events* executing *actions*, which are given by their node labeling.

Consider the graph from Fig. 3.4a. It is contained in $\mathbb{DAG}_H(\{a, b\})$ (recall that we may omit "$-$" if there are no edge labelings), as its edge relation is a minimal one to generate some partial order. In contrast, the graph from Fig. 3.4b, though it is contained in $\mathbb{DAG}(\{a, b\})$, is not minimal in this sense, as there is an edge from u to w that is already implicitly present in terms of (u, v) and (v, w). Thus, Fig. 3.4b illustrates a dag that is not contained in $\mathbb{DAG}_H(\{a, b\})$. Finally, the graph from Fig. 3.4c is not even contained in $\mathbb{DAG}(\{a, b\})$, as it is not acyclic. However, it is a member of $\mathbb{DG}(\{a, b\})$.

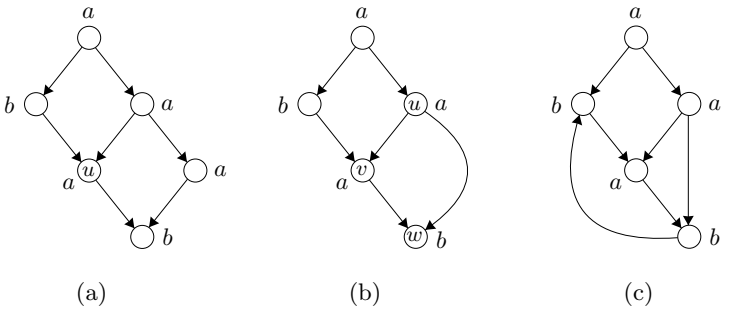


Fig. 3.4. Graphs over $(\{a, b\}, -)$

For $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell\in C}, \lambda) \in \mathbb{DAG}(\Sigma, C)$ and $u \in V$, let $\mathcal{G} \Downarrow u$ stand for the *downwards closure* of $\mathcal{G}$ with respect to u, i.e., for $(V', \{\lhd'_\ell\}_{\ell\in C}, \lambda') \in \mathbb{DAG}(\Sigma, C)$ where $V' = \{v \in V \mid v \leq u\}$, $\lhd'_\ell = \lhd_\ell \cap (V' \times V')$ for any $\ell \in C$, and $\lambda' = \lambda_{|V'}$, i.e., λ' is the restriction of λ to the nodes from V'. The *strict downwards closure* of $\mathcal{G}$ with respect to u, denoted by $\mathcal{G} \downarrow u$, is obtained in the same way as the downwards closure, now taking $V' = \{v \in V \mid v < u\}$ as a starting point. To exemplify downwards closure, consider once again the

graph $\mathcal{G}$ from Fig. 3.4a: the downwards closure of $\mathcal{G}$ with respect to u is given by Fig. 3.5a, its strict downwards closure is depicted in Fig. 3.5b.

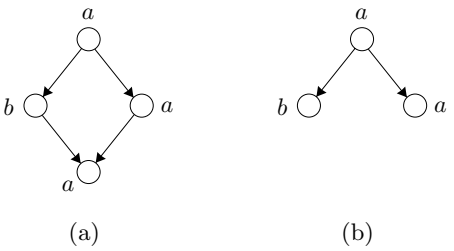


Fig. 3.5. The downwards closure of a graph

3.6 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 dags and 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 $n \in \mathbb{N}_{\geq 1}$, let $[n]$ denote the set $\{1, \ldots, n\}$.

Definition 3.26 (Picture). *A* picture *over Σ is a graph*

$$([n] \times [m], S_1, S_2, \lambda) \in \mathbb{DAG}_H(\Sigma, \{1, 2\})$$

over $(\Sigma, \{1, 2\})$ 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, Fig. 3.6 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 readability.

Though *tiling systems*, where a tiling is a square of length two rather than a graph around a center (see [38] for an overview), is arguably the more natural automata model for pictures, we stick to the familiar terrain of graph acceptors. As with traces and message sequence charts (cf. Chaps. 6 and 8), graph acceptors yield different results than the general case if they are applied to pictures rather than arbitrary graphs.

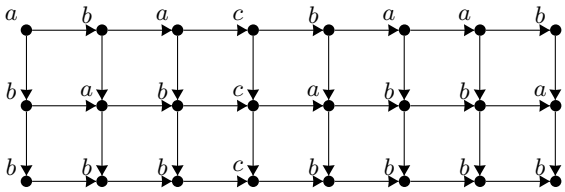


Fig. 3.6. A picture over $\{a, b, c\}$

Lemma 3.27 ([92]).

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

Again, the proof of Lemma 3.27 is a simple reduction involving a blow-up in the number of states of the graph acceptor. It is left to the reader as an exercise.

Theorem 3.28 ([92]). *In general, $\mathcal{FO}[\leq]_{\mathbb{P}(\Sigma)}$ and $\mathcal{EMSO}_{\mathbb{P}(\Sigma)}$ are incomparable with respect to 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 $\mathrm{FO}[\leq]$-definable relative to Σ^*, 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* $\mathcal{GCH}$ of some picture $\mathcal{C} \in \mathbb{P}(\{c\})$ consisting of one single c-labeled column and pictures $\mathcal{G}, \mathcal{H} \in \mathbb{P}(\{a, b\})$ such that the sets of different column labelings of $\mathcal{G}$ and $\mathcal{H}$ coincide. A picture that belongs to L is depicted in Fig. 3.6. Its unique division into $\mathcal{G}$, $\mathcal{C}$, and $\mathcal{H}$ as postulated above and illustrated in Fig. 3.7 gives rise to the set of column words $\{abb, bab\}$, which represents both $\mathcal{G}$ and $\mathcal{H}$. In fact, L is $\mathrm{FO}[\leq]_{\mathbb{P}(\Sigma)}$-definable.

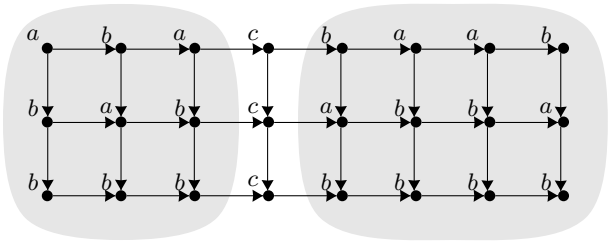


Fig. 3.7. Dividing a picture over $\{a, b, c\}$

A corresponding sentence just has to require that, for any node u on the first row, there has to be a suitable counterpart v that is also located on the first row, but on the opposite side of the c-labeled column. Moreover, the columns below u and v have to coincide. The latter can be easily formalized using the predicates $\leq_1$ and $\leq_2$ (recall that, for example, $\leq_2$ 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 3.24 and Lemma 3.27, there is a graph acceptor $\mathcal{B} = (Q, R, \delta, Occ)$ 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 $\mathcal{GCH}$, all the information carried from $\mathcal{G}$ to $\mathcal{H}$ is already present in the sequence of states associated to the single column $\mathcal{C}$. 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 $\mathcal{G}'\mathcal{CH}'$ for some $\mathcal{G}', \mathcal{H}' \in \mathbb{P}(\{a, b\})$ that induce different sets of column words. □

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 $\mathcal{G}(n, m) := ([n] \times [m], S_1, S_2) \in \mathbb{DAG}_H(-, \{1, 2\})$ 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, moreover, $((i, j), (i, j + 1)) \in ([n] \times [m])^2$, respectively. By $\mathbb{GR}$, we denote the set of all grids. The (3,5)-grid is depicted in Fig. 3.8.

A relation $\mathcal{R} \subseteq \mathbb{N}_{\geq 1} \times \mathbb{N}_{\geq 1}$ may be represented by the grid language $\{\mathcal{G}(n, m) \mid (n, m) \in \mathcal{R}\}$. As a unary function $f : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ can be considered as a binary relation, we accordingly define the *grid language* $\mathcal{G}(f)$ of f to be the set $\{\mathcal{G}(n, f(n)) \mid n \in \mathbb{N}_{\geq 1}\}$.

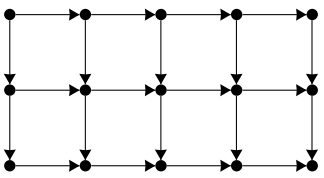


Fig. 3.8. The (3,5)-grid

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 3.29 ([66]). *The monadic quantifier-alternation hierarchy over* $\mathbb{GR}$ *is infinite.*

The next result directly follows from Lemma 3.27.

Corollary 3.30.

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

Though, with respect to 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 3.22). This fact is witnessed by the set L_n of *n-supergrids* for some $n \geq 4$ (cf. [92]). 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 the set of graphs over $(-, \{1, 2\})$, 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 $L_{\mathbb{DG}(-,\{1,2\})}(\mathcal{B}) = L_n$ and consider ρ to be an accepting run of $\mathcal{B}$ on some $\mathcal{G} \in L_n$. If $\mathcal{G}$ 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 with respect to the partial order induced by $\mathcal{G}$. Moreover, suppose the nodes of $\mathcal{G}$ that belong to the corresponding two sphere centers to be u and v and assume that their (only) outgoing edges lead to u' and v', respectively. From $\mathcal{G}$, we obtain another graph if we remove the edges (u, u') and (v, v') and, instead, add (u, v') and (v, u'). The resulting graph, though it is contained in $\mathbb{DAG}(-, \{1, 2\})$ and in $L_{\mathbb{DAG}(-,\{1,2\})}(\mathcal{B})$, is no longer a supergrid.

3.7 Bibliographic Notes

Standard logic references are [30, 39]. First-order and monadic second-order logic over words and graphs and their relation to finite automata are moreover studied in [94], which also provides some background of the theorem of Hanf. In this regard, we also recommend [60], a comprehensive textbook on finite model theory. Graph acceptors, whose logical characterization is based on the theorem of Hanf, are introduced in [91]. However, the notation of a graph acceptor that we adopted in this book is provided in [92, 93], which also give a broad overview of its applications and its relation to (existential) monadic second-order logic over dags, posets, grids, and pictures.

4

Words and Finite Automata

In this chapter, we recall the notion of finite automata, which we consider to be a sequential model without any communication. Actions are processed in a linear manner and therefore arranged as words.

4.1 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. In the following, let Σ be an alphabet.

Definition 4.1 (Word). *A* word *over Σ is a structure $\underline{w} = (\{1, \ldots, n\}, \lhd, \lambda)$, $n \in \mathbb{N}$, where $1, \ldots, n$ are the* letter positions *of $\underline{w}$, $\lhd$ 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. Actually, $\mathbb{W}(\Sigma)$ is simply $\mathbb{DAG}_H(\Sigma, -)$ restricted to graphs $(V, \lhd, \lambda)$ 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 $a_1 \ldots a_n \in \Sigma^*$ where $a_i = \lambda(i)$ for each $i \in \{1, \ldots, n\}$. For example, $(\{1,2,3\}, \{(1,2),(2,3)\}, \{1 \mapsto a, 2 \mapsto b, 3 \mapsto a\}) \in \mathbb{W}(\{a,b\})$ is equated with the string $aba \in \{a,b\}^*$. Recall that the empty word, which corresponds to the structure $(\emptyset, \emptyset, \emptyset)$, is denoted by ε. It appears as the unit word in the free monoid $\mathbb{W}$. Recognizability and rationality coincide in finitely generated free monoids, which is, as commonly known, Kleene's theorem.

Given a word $\underline{w} = (\{1, \ldots, n\}, \lhd, \lambda) \in \mathbb{W}$ with $n \geq 1$, we denote by $\mathit{first}(\underline{w})$ and $\mathit{last}(\underline{w})$ the events 1 and n, respectively.

Theorem 4.2 (Kleene [53]).

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

Exercise 4.3. Let $\Sigma = \{a, b\}$. Provide rational expressions $\alpha_1, \alpha_2, \alpha_3$ of $\mathbb{W}(\Sigma)$ such that the following hold:

$L(\alpha_1) = \{\underline{w} \in \mathbb{W}(\Sigma) \mid \underline{w} \text{ does not end with } ab\}$,
$L(\alpha_2) = \{\underline{w} \in \mathbb{W}(\Sigma) \mid ab \text{ occurs in } \underline{w} \text{ exactly once}\}$,
$L(\alpha_3) = L(\alpha_1) \cup L(\alpha_2)$.

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

- $\lambda(x) = a$,
- $x \lhd y$,
- $x \in X$, and
- $x = y$.

The definition of their semantics arises from the general case of graphs (cf. Chap. 3).

Exercise 4.4. Describe each of the following word languages over $\Sigma = \{a, b\}$ by an $\mathrm{MSO}(\Sigma, -)$-sentence relative to $\mathbb{W}(\Sigma)$:

(a) $\{a\underline{w}a \mid \underline{w} \in \mathbb{W}(\Sigma)\}$,
(b) $\{\underline{w} \in \mathbb{W}(\Sigma) \mid |\underline{w}|_b > 3\}$,
(c) $\{\underline{w} \in \mathbb{W}(\Sigma) \mid |\underline{w}| \text{ is even}\}$.

4.2 Finite Automata

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

Definition 4.5 (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 $\underline{w} = (\{1, \ldots, n\}, \lhd, \lambda) \in \mathbb{W}(\Sigma)$ is a mapping $\rho : \{0, 1, \ldots, n\} \to S$ such that $\rho(0) = s^{in}$ and, for any $i \in \{1, \ldots, n\}$, $(\rho(i-1), \lambda(i), \rho(i)) \in \Delta$. We call ρ *accepting* if $\rho(n) \in F$. The *language* of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set $\{\underline{w} \in \mathbb{W} \mid \text{there is an accepting run of } \mathcal{A} \text{ on } \underline{w}\}$. Note that ε is contained in

$L(\mathcal{A})$ if (and only if) $s^{in} \in F$. We call $\mathcal{A}$ *deterministic* if, for any $s, s_1, s_2 \in S$ and $a \in \Sigma$, $\{(s, a, s_1), (s, a, s_2)\} \subseteq \Delta$, implies $s_1 = s_2$. In that case, $\mathcal{A}$ provides for any word $\underline{w} \in \mathbb{W}(\Sigma)$ at most one run on $\underline{w}$. Note that, often, one requires a deterministic automaton to provide *exactly* one run on any word, which, however, can be easily achieved by adding a *sink* state, i.e., a nonaccepting state that goes to itself on every action from Σ.

By $\mathcal{FA}(\Sigma)$ (det-$\mathcal{FA}(\Sigma)$), we denote the class of word languages that are recognized by some (deterministic, respectively) finite automaton over Σ. If Σ is clear from the context, we usually write $\mathcal{FA}$ and det-$\mathcal{FA}$.

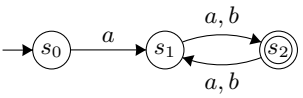


Fig. 4.1. A finite automaton

Example 4.6. The finite automaton $\mathcal{A} = (\{s_0, s_1, s_2\}, \Delta, s_0, \{s_2\})$ over $\{a, b\}$ with $\Delta = \{(s_0, a, s_1), (s_1, a, s_2), (s_1, b, s_2), (s_2, a, s_1), (s_2, b, s_1)\}$ is depicted in Fig. 4.1. In particular, the initial state and the only final state are marked by an ingoing arrow without source state and by a second circle, respectively. Observe that $\mathcal{A}$ is deterministic and that its language is $L((aa + ab) \cdot (aa + ab + ba + bb)^*)$, i.e., the set of words that start with an a and are of even length.

Theorem 4.7.

$$\mathcal{FA} \;=\; \text{det-}\mathcal{FA}$$

The construction of a deterministic finite automaton from a nondeterministic one is based on the well-known *power-set construction*. Its basic idea is to simulate several runs deterministically by collecting all the possible states in which the nondeterministic model might be (of course, there are finitely many of them). For details, we refer to [48].

Exercise 4.8. Suppose $\Sigma = \{a, b\}$. Determine directly (i.e., without going over nondeterministic automata) deterministic finite automata $\mathcal{A}_1, \ldots, \mathcal{A}_4$ over Σ such that the following hold:

$L(\mathcal{A}_1) = \{\underline{w} \in \mathbb{W}(\Sigma) \mid \underline{w} \notin \{ab, ba\}^*\}$,
$L(\mathcal{A}_2) = \{\underline{w} \in \mathbb{W}(\Sigma) \mid \underline{w} \notin \{ab\}^* \text{ and } w \notin \{ba\}^*\}$,
$L(\mathcal{A}_3) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$,
$L(\mathcal{A}_4) = L(\mathcal{A}_1) \cup L(ab^*a)$.

A finite automaton gives rise to a $\mathbb{W}$-automaton and vice versa (see Exercise 4.16 below).

Theorem 4.9.

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

Moreover, finite automata can be characterized in terms of MSO, which is the famous theorem of Büchi and Elgot.

Theorem 4.10 (Büchi, Elgot [20, 32]).

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

Proof. "$\subseteq$": Suppose $\mathcal{A} = (S, \Delta, s^{in}, F)$ to be a finite automaton over Σ, say, with state set $S = \{s_0, \ldots, s_k\}$ where $s^{in} = s_0$. Then, for any word $\underline{w} = (\{1, \ldots, n\}, \lhd, \lambda) \in \mathbb{W}(\Sigma)$ with $\underline{w} \neq \varepsilon$, we have $\underline{w} \in L(\mathcal{A})$ iff

$$\begin{aligned}
\underline{w} \models\; & \exists X_0 \ldots \exists X_k \\
& \Big[\; \mathrm{partition}(X_0, \ldots, X_k) \\
& \wedge\, \forall x (\mathit{last}(x) \rightarrow \textstyle\bigvee_{s_i \in F} x \in X_i) \\
& \wedge\, \forall x \forall y \left(x \lhd y \rightarrow \textstyle\bigvee_{(s_i, a, s_j) \in \Delta} (x \in X_i \wedge \lambda(y) = a \wedge y \in X_j) \right) \\
& \wedge\, \forall x \left(\mathit{first}(x) \rightarrow \textstyle\bigvee_{(s_0, a, s_i) \in \Delta} (\lambda(x) = a \wedge x \in X_i) \right) \Big]
\end{aligned}$$

Here, the predicates $\mathit{first}(x)$ and $\mathit{last}(x)$ are used to abbreviate $\neg\exists y(y \lhd x)$ and $\neg\exists y(x \lhd y)$, 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 has to add a clause $\exists x \mathit{first}(x)$, as, otherwise, ε will be included in the language of the above formula.

"$\supseteq$": So let us construct from an arbitrary $\mathrm{MSO}(\Sigma, -)$-sentence ψ a finite automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L_{\mathbb{W}(\Sigma)}(\psi)$. According to Lemma 3.3, it is sufficient to give an inductive translation of an $\mathrm{MSO}_0(\Sigma, -)$-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 first 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. $\square$

The proof of Theorem 4.10 allows us to effectively construct from any given MSO-sentence an equivalent EMSO-sentence.

Corollary 4.11.

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

Remark 4.12. Unfortunately, transforming an MSO formula into an equivalent finite automaton has nonelementary complexity. More precisely, there is no translation of MSO formulas φ into finite automata $\mathcal{A}$ such that the number of states of $\mathcal{A}$ is bounded by an elementary function, i.e., by an iterated exponential of the form

$$2^{2^{\cdot^{\cdot^{\cdot 2^{|\varphi|}}}}}$$

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

Proposition 4.13.

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

Proof. While the first strict inclusion is witnessed by the language of the rational expression $a^*ba^*ca^*$, which is $\mathrm{FO}[\leq]_{\mathbb{W}}$- but not $\mathrm{FO}_{\mathbb{W}}$-definable, the set of words of even length turns out to be $\mathrm{EMSO}_{\mathbb{W}}$- but not $\mathrm{FO}[\leq]_{\mathbb{W}}$-definable. See [92, 94] for further details. □

Exercise 4.14. Show that $L(a^*ba^*ca^*) \notin \mathcal{FO}_{\mathbb{W}(\{a,b,c\})}$.

An important concept of dags (and posets) is their characterization in terms of *linear extensions* or *linearizations*, which establish a relationship between dags and words. So let $\mathcal{G} = (V, \{\lhd_\ell\}_{\ell\in C}, \lambda)$ be a graph from $\mathbb{DAG}(\Sigma, C)$. A word $(V', \lhd', \lambda') \in \mathbb{W}(\Sigma)$ is called a *linearization* of $\mathcal{G}$ if $V' = V$, $\lhd'$ is the covering relation of some total order containing $\lhd^*$, and $\lambda' = \lambda$. The set of linearizations of $\mathcal{G}$ is denoted by $\mathit{Lin}(\mathcal{G})$. This notion is extended to sets L of graphs according to $\mathit{Lin}(L) := \bigcup_{\mathcal{G}\in L} \mathit{Lin}(\mathcal{G})$. For example, *abaa* and *aaba* are the only linearizations of the graph depicted in part (a) of Fig. 3.5 on page 31. 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 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 4.15.

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

Proof. The second equality directly follows from Theorem 3.24, Theorem 4.10, and Corollary 4.11. The proof of the first equality is left to the reader as an exercise (cf. Exercise 4.16). For this, one has to verify that the transitions of a finite automaton can be simulated using 1-spheres. In particular, initial/final transitions are mimicked by 1-spheres whose sphere center has no predecessor/successor, respectively. □

Exercise 4.16. Give direct transformations for the following directions:

(a) from finite automata to $\mathbb{W}$-automata,
(b) from $\mathbb{W}$-automata to finite automata,
(c) from graph acceptors with 1-spheres to finite automata,
(d) from finite automata to graph acceptors with 1-spheres.

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 3.23).

Lemma 4.17 ([92]).

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

4.3 Summary

Some closure and expressiveness properties of finite automata are summarized in Table 4.1. In fact, the class $\mathcal{FA}$ is closed under union, intersection, and complementation. Moreover, the deterministic model of a finite automaton already achieves the expressiveness of its general model. Finite automata are expressively equivalent to EMSO logic, which, in turn, has the same power as full MSO logic. Finally, it is obviously decidable if a given finite automaton recognizes the empty language.

Table 4.1. Closure and expressiveness properties of finite automata

$\cup$	$\cap$	$\bar{\cdot}$	det	EMSO	MSO	Empt.
✓	✓	✓	=	=	=	✓

4.4 Bibliographic Notes

For a general and accessible introduction to formal languages and finite automata, see [48]. Therein, the reader may find basic transformations of finite automata such as complementation and determinization. The equivalence between finite automata and monadic second-order logic over words is summarized, for example, in [60] and [94]. First-order and monadic second-order logics as well as graph acceptors over words are considered in [92] and [94].

5

Dags and Asynchronous Cellular Automata

In this chapter, we present asynchronous cellular automata (with types) as a general model of a distributed system. It subsumes the models of finite automata, asynchronous automata, and communicating finite-state machines, as introduced in Chaps. 4, 6, and 8, respectively. Nevertheless, it allows a characterization in terms of EMSO logic. The underlying domain comprises dags over a distributed alphabet, which describes the dependencies of actions provided by the system.

5.1 $(\widetilde{\Sigma}, C)$-Dags

In this section, 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 $\mathbb{DAG}$ of directed acyclic graphs, which generate partial orders in a natural manner. Recall that 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 suffices to consider only those graphs that represent totally ordered sets or words (cf. Chap. 4). 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*. 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.

We fix a nonempty finite set Ag of at least two *agents*, a *distributed alphabet* $\widetilde{\Sigma}$, which is a tuple $(\Sigma_i)_{i\in Ag}$ of (not necessarily disjoint) alphabets Σ_i, and an alphabet C. In the following, let Σ stand for $\bigcup_{i\in Ag}\Sigma_i$, the set of *actions*. Elements from Σ_i are understood to be actions that are performed by agent i. So let, for $a \in \Sigma$, $loc(a) := \{i \in Ag \mid a \in \Sigma_i\}$ denote the set of agents that are involved in a. Having this in mind, we say that actions a and b are *independent* and write $a\ I_{\widetilde{\Sigma}}\ b$ if there is no common agent that controls both of them, i.e., if $loc(a) \cap loc(b) = \emptyset$. Otherwise, we say a and b are *dependent*, writing $a\ D_{\widetilde{\Sigma}}\ b$. Observe that $D_{\widetilde{\Sigma}}$ is a reflexive and symmetric binary relation on Σ. Such a relation is called a *dependence relation* over Σ. Moreover, we call the pair $(\Sigma, D_{\widetilde{\Sigma}})$ a *dependence alphabet*.

Agents may communicate with one another. They may do this by executing common actions simultaneously or via message exchange through channels. So let us denote by $Ch(Ag)$ (or just Ch) the set $\{(i,j) \in Ag \times Ag \mid i \neq j\}$ of *channels*. Thus, we assume that a "real" message exchange takes place between distinct processes only.

We now introduce the models representing the behavior of a system of communicating agents. In doing so, we combine the standard models of [29] and [55].

Definition 5.1 (Lo-Dag). *A* lo-dag *over the pair* $(\widetilde{\Sigma}, C)$ *is a structure* $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda) \in \mathbb{DAG}(\Sigma, C)$ *such that, for any* $i \in Ag$, $\lambda^{-1}(\Sigma_i)$ *is totally ordered by* $\leq$.

Thus, the only restriction imposed by a lo-dag compared with a dag is the totally ordered behavior of a single process, which is thus assumed to be sequential. If, in addition, we restrict the communication between agents in the way that messages (edges) of equal type cannot cross, we obtain the class of $(\widetilde{\Sigma}, C)$-dags, which will be the main model considered in this chapter.

Definition 5.2 ($(\widetilde{\Sigma}, C)$-Dag). *A* $(\widetilde{\Sigma}, C)$-dag *is a lo-dag* $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda)$ *over* $(\widetilde{\Sigma}, C)$ *such that, for any* $\ell \in C$ *and any* $(u,v), (u',v') \in \lhd_\ell$ *with* $\lambda(u) = \lambda(u')$ *and* $\lambda(v) = \lambda(v')$, *we have* $u \leq u'$ *iff* $v \leq v'$.

We conclude that, in a $(\widetilde{\Sigma}, C)$-dag $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda)$, for any $u \in V$, $\ell \in C$, and $a \in \Sigma$, there is at most one vertex $v \in V$ such that both $u \lhd_\ell v$ $(v \lhd_\ell u)$ and $\lambda(v) = a$. If we require that *any* two messages between two agents must not cross, we deal with the class of fifo-dags over $(\widetilde{\Sigma}, C)$.

Definition 5.3 (Fifo-Dag). *A* fifo-dag *over the pair* $(\widetilde{\Sigma}, C)$ *is a lo-dag* $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda)$ *over* $(\widetilde{\Sigma}, C)$ *such that, for any* $(u,v), (u',v') \in \lhd$ *with* $\lambda(u)\ D_{\widetilde{\Sigma}}\ \lambda(u')$ *and* $\lambda(v)\ D_{\widetilde{\Sigma}}\ \lambda(v')$, *we have* $u \leq u'$ *iff* $v \leq v'$.

Thus, fifo-dags correspond to a fifo architecture where messages (u, v) and (u', v') between agents i and j are received by j in terms of v and v' in the order they have been sent by i in terms of u and u', respectively. The sets of lo-dags over $(\widetilde{\Sigma}, C)$, $(\widetilde{\Sigma}, C)$-dags, and fifo-dags over $(\widetilde{\Sigma}, C)$ will be denoted by $\mathbb{DAG}_{lo}(\widetilde{\Sigma}, C)$, $\mathbb{DAG}(\widetilde{\Sigma}, C)$, and $\mathbb{DAG}_{\rightrightarrows}(\widetilde{\Sigma}, C)$, respectively. Recall that, in contrast, $\mathbb{DAG}(\Sigma, C)$ was used to denote the set of dags over (Σ, C) in general.

As usual, if C is a singleton and therefore negligible, we may write, for example, $\mathbb{DAG}(\widetilde{\Sigma}, -)$ or just $\mathbb{DAG}(\widetilde{\Sigma})$ instead of $\mathbb{DAG}(\widetilde{\Sigma}, C)$, and we may speak of $\widetilde{\Sigma}$-dags or fifo-dags over $\widetilde{\Sigma}$.

Remark 5.4. $\mathbb{DAG}_{\rightrightarrows}(\widetilde{\Sigma}, C) \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C) \subseteq \mathbb{DAG}_{lo}(\widetilde{\Sigma}, C) \subseteq \mathbb{DAG}(\Sigma, C)$

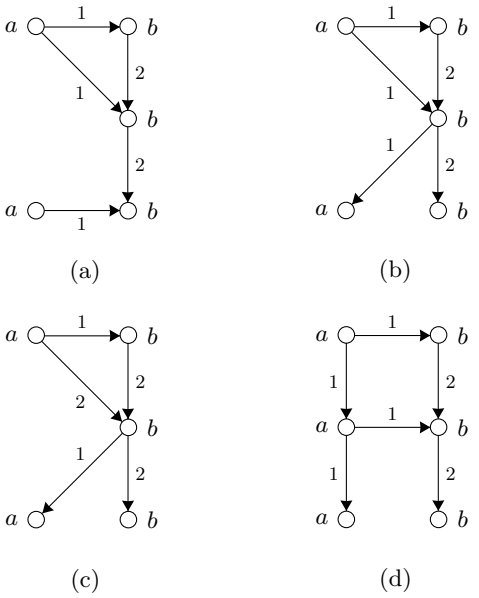


Fig. 5.1. Some dags over $(\{a, b\}, \{1, 2\})$

Example 5.5. Suppose $\widetilde{\Sigma} = (\{a\}, \{b\})$ and $C = \{1, 2\}$. Figure 5.1a depicts a dag over (Σ, C) that is not a lo-dag over $(\widetilde{\Sigma}, C)$, because the a-labeled nodes are not ordered with respect to $\leq$. Figure 5.1b illustrates a lo-dag over $(\widetilde{\Sigma}, C)$ that is not a $(\widetilde{\Sigma}, C)$-dag, whereas the dag from Fig. 5.1c is a $(\widetilde{\Sigma}, C)$-dag that, however, is not a fifo-dag. Finally, Fig. 5.1d depicts a fifo-dag over $(\widetilde{\Sigma}, C)$.

Example 5.6. Two fifo-dags over $(\{a\}, \{b, c\}, \{c, d\})$ (thus, without edge labelings) are depicted in Fig. 5.2. Observe that the right-hand figure shows the 1-sphere of the other dag around the d-labeled node.

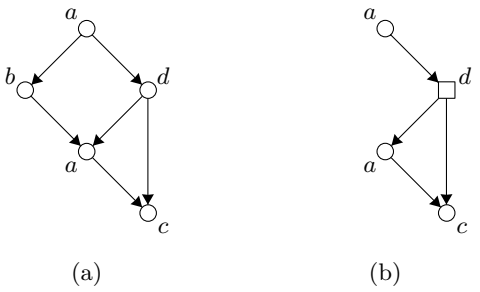


Fig. 5.2. A fifo-dag over $(\{a\}, \{b, c\}, \{c, d\})$ and a 1-sphere

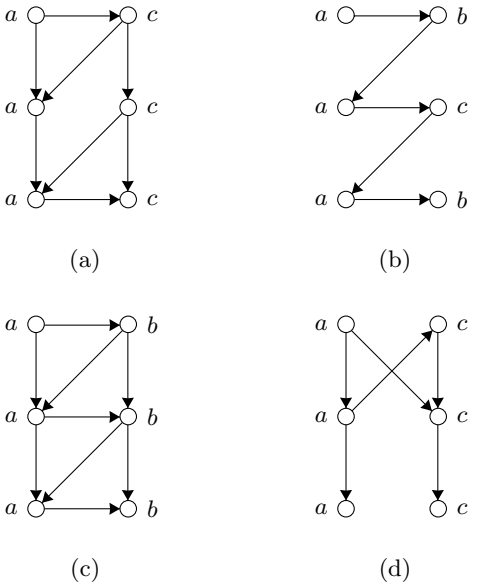


Fig. 5.3. Some lo-dags over $((\{a, b\}, \{b, c\}), -)$

Example 5.7. Consider Fig. 5.3. If we suppose $\widetilde{\Sigma}$ to be $(\{a, b\}, \{b, c\})$, parts (a) and (b) both depict a fifo-dag over $\widetilde{\Sigma}$. Part (c) illustrates a $\widetilde{\Sigma}$-dag that is not a fifo-dag. Finally, the remaining dag, from Fig. 5.3d, is a lo-dag over $\widetilde{\Sigma}$ but not a $\widetilde{\Sigma}$-dag.

Remark 5.8. Any class $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$ has bounded degree.

Exercise 5.9. Depending on $\widetilde{\Sigma}$ and C, determine the lowest natural number B such that the degree of $\mathbb{DAG}(\widetilde{\Sigma}, C)$ is bounded by B. Is $\mathbb{DAG}_{lo}(\widetilde{\Sigma}, C)$ bounded, too?

Given $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda) \in \mathbb{DAG}_{lo}(\widetilde{\Sigma}, C)$, let us introduce the following abbreviations: For $i \in Ag$, V_i shall denote the set of nodes $u \in V$ such that $\lambda(u) \in \Sigma_i$. Accordingly, given $a \in \Sigma$, V_a is the set of nodes $u \in V$ such that $\lambda(u) = a$. Let $u, v \in V$. We write $u \lhd_{\mathrm{c}} v$ if both $\lambda(u)\ I_{\widetilde{\Sigma}}\ \lambda(v)$ and $u \lhd v$ (thus, u and v communicate by means of a channel as they have no agent in common that could have synchronized them). If, in addition, u and v make use of some channel $(i, j) \in Ch$, i.e., if $u \lhd_{\mathrm{c}} v$, $u \in V_i$, and $v \in V_j$, then we write $u \lhd_{(i,j)} v$. Note that, even if we deal with a fifo-dag, we may have $u \lhd_{ch} v$ and $u \lhd_{ch'} v$ at the same time for distinct channels $ch, ch' \in Ch$. Otherwise, sequential progress of an agent $i \in Ag$ is reflected by relations $\lhd_i := \lhd \cap (V_i \times V_i)$ and $\leq_i := \leq \cap (V_i \times V_i)$, which is a total order. Moreover, $<_i$ is defined in the obvious manner. We observe that the statement $u \lhd_i v$ may hold just as well as $u \lhd_j v$ for distinct agents i and j. Note that it will always be clear if the index of the symbol $\lhd$ is meant to be a channel, an agent, or a symbol from C. For $u \in V$ and $i \in Ag$, we say that u is Σ_i-maximal if $u \in V_i$ and there is no $v \in V_i$ such that $u < v$. There is at most one Σ_i-maximal vertex.

In the following, we will focus on $(\widetilde{\Sigma}, C)$-dags rather than lo- and fifo-dags. Similarly to the general case, we write $\mathbb{DAG}_H(\widetilde{\Sigma}, C)$ to denote $\mathbb{DAG}(\widetilde{\Sigma}, C) \cap \mathbb{DAG}_H(\Sigma, C)$. Let $(V, \{\lhd_\ell\}_{\ell\in C}, \lambda)$ be a $(\widetilde{\Sigma}, C)$-dag and let $u \in V$. We denote by $\mathrm{Read}(u) := \{(a, \ell) \in \Sigma \times C \mid$ there is some $v \in V$ such that $v \lhd_\ell u$ and $\lambda(v) = a\}$ the *read domain* of u and, given $(a, \ell) \in \mathrm{Read}(u)$, let (a, ℓ)-pred(u) be the unique vertex v such that both $v \lhd_\ell u$ and $\lambda(v) = a$. Accordingly, let $\mathrm{Write}(u) := \{(a, \ell) \in \Sigma \times C \mid$ there is some $v \in V$ such that $u \lhd_\ell v$ and $\lambda(v) = a\}$ be the *write domain* of u and, for $(a, \ell) \in \mathrm{Write}(u)$, (a, ℓ)-succ(u) denote the unique vertex v such that both $u \lhd_\ell v$ and $\lambda(v) = a$. If C is a singleton, $\mathrm{Write}(u)$ and $\mathrm{Read}(u)$ might be considered to be subsets of Σ, which allows us to write simply a-pred(u) and a-succ(u), respectively.

A fundamental notion concerning $(\widetilde{\Sigma}, C)$-dags is that of projections onto agents, which describes the totally ordered action sequence of one single agent. Such an action sequence gives rise to a word. For $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell\in C}, \lambda) \in \mathbb{DAG}(\widetilde{\Sigma}, C)$ and $i \in Ag$, we denote by $\mathcal{D} \restriction i$ the *projection* $(V', \lhd', \lambda') \in \mathbb{W}(\Sigma_i)$ of $\mathcal{D}$ onto i where

- $V' = V_i$,
- $\lhd'$ is the covering relation of $\leq_i$, and
- $\lambda' = \lambda_{|V'}$ (i.e., λ' is the restriction of λ to V').

For example, the words aa, bc, and dc are the projections $\mathcal{D} \restriction 1$, $\mathcal{D} \restriction 2$, and $\mathcal{D} \restriction 3$ of the $(\{a\}, \{b, c\}, \{c, d\})$-dag $\mathcal{D}$ from Fig. 5.2a onto 1, 2, and 3, respectively.

Example 5.10 (Mazurkiewicz Traces (1)). An M^+-*trace* over $\widetilde{\Sigma}$ is a structure $(V, \{\lhd_\ell\}_{\ell\in 2^{Ag}}, \lambda) \in \mathbb{DAG}(\widetilde{\Sigma}, 2^{Ag})$ such that

- $\lhd = \bigcup_{i\in Ag} \lessdot_i$, and
- for any $(u, v) \in \lhd$ and $\ell \in 2^{Ag}$, $u \lhd_\ell v$ iff $\ell = \{i \in Ag \mid u \lessdot_i v\}$.

Recall that, hereby, $\lessdot_i$ is the covering relation of the total order $\leq_i$. Thus, the labeling of an edge between nodes u and v (which can be assumed to be nonempty) tells us which agents execute u and v consecutively. The set of M^+-traces over $\widetilde{\Sigma}$ is denoted by $\mathbb{TR}^+(\widetilde{\Sigma})$. Figure 5.4a depicts an M^+-trace over $(\{a,b,d\},\{a,b,e\},\{a,b\})$ (say, with $Ag=\{1,2,3\}$). The dag from Fig. 5.4b is clearly not an M^+-trace. Apart from the edge labelings, it lacks the mandatory edge between the a and the b-labeled node. However, the latter dag is just a different view of the former. Its edge relation allows us to access the history of a node only with respect to $\lessdot$ (cf. Example 5.11).

Example 5.11 (Mazurkiewicz Traces (2)). An M^-*-trace* over $\widetilde{\Sigma}$ is a structure $(V,\lhd,\lambda)\in\mathbb{DAG}_H(\widetilde{\Sigma},-)$ such that, for any $u,v\in V$, $u\lhd v$ implies $\lambda(u)\ D_{\widetilde{\Sigma}}\ \lambda(v)$. Note that, as we consider a subclass of $\mathbb{DAG}_H(\widetilde{\Sigma},-)$, $\lhd$ and $\lessdot$ coincide. The set of M^--traces over $\widetilde{\Sigma}$ is denoted by $\mathbb{TR}^-(\widetilde{\Sigma})$. Figure 5.4b depicts an M^--trace over $(\{a,b,d\},\{a,b,e\},\{a,b\})$, whereas the dag from Fig. 5.4a (neglecting the edge labelings) is not an M^--trace over $(\{a,b,d\},\{a,b,e\},\{a,b\})$. In particular, it is not a Hasse diagram.

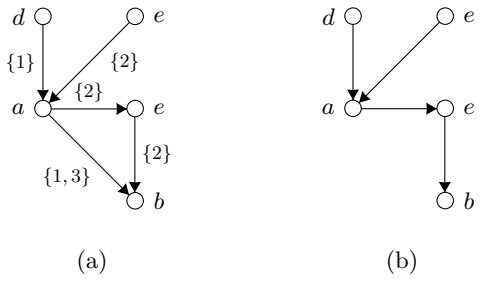


Fig. 5.4. An M^+-trace and an M^--trace over $(\{a,b,d\},\{a,b,e\},\{a,b\})$

Example 5.12 (Message Sequence Charts). Messages may be exchanged between the agents by performing send and receive actions. So set, for an agent $i\in Ag$, Γ_i to be $\{i!j\mid j\in Ag\setminus\{i\}\}\cup\{i?j\mid j\in Ag\setminus\{i\}\}$, the set of *(communication) actions* of agent i. The action $i!j$ is to be read as "i sends a message to j", while $j?i$ is the complementary action of receiving a message sent from i to j. Let Γ stand for the union of the (disjoint) Γ_i and set $\widetilde{\Gamma}$ to be the distributed alphabet $(\Gamma_i)_{i\in Ag}$. A *message sequence chart* (MSC) over Ag is a $\widetilde{\Gamma}$-dag $(V,\lhd,\lambda)$ (which is thus contained in $\mathbb{DAG}(\widetilde{\Gamma},-)$) such that

1. for any $i\in Ag$, $\lhd_i$ is the cover relation of $\leq_i$,
2. for any $(u,v)\in\lhd_{\mathrm{c}}$, $\lambda(u)$ is a send action and $\lambda(v)$ is its complementary receive, and
3. for any $u\in V$, there is $v\in V$ satisfying either $u\lhd_{\mathrm{c}} v$ or $v\lhd_{\mathrm{c}} u$.

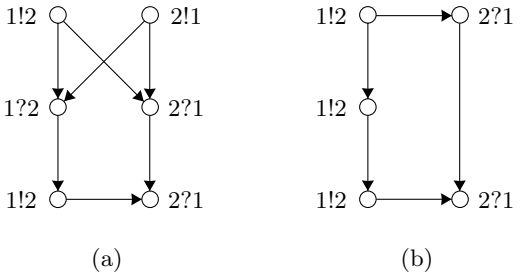


Fig. 5.5. An MSC and an LMSC over $\{1, 2\}$, which are both fifo-dags over $(\{1!2, 1?2\}, \{2!1, 2?1\})$

We denote by $\mathbb{MSC}(Ag)$ the set of MSCs over Ag. Note that, by the definition of a $\widetilde{\Gamma}$-dag, an MSC behaves in a fifo manner, neglecting overtaking of messages of equal type. If we do not require a send vertex to be equipped with a corresponding receive, we obtain the class of (potentially) *lossy* MSCs (LMSCs) over Ag, which is a superset of $\mathbb{MSC}(Ag)$ and shall be denoted by $\mathbb{LMSC}(Ag)$. In particular, we obtain $\mathbb{MSC}(Ag) \subseteq \mathbb{LMSC}(Ag) \subseteq \mathbb{DAG}_{\rightrightarrows}(\widetilde{\Gamma})$. Figure 5.5 illustrates an MSC over $\{1, 2\}$ and, respectively, an LMSC over $\{1, 2\}$, which is not an MSC. Note that $\mathbb{MSC}(Ag)$ might be defined relative to $\mathbb{LMSC}(Ag)$ by the $\mathrm{FO}(\Gamma)$-sentence

$$\forall x \bigwedge_{(i,j)\in Ch(Ag)} (\lambda(x) = i!j \rightarrow \exists y(x \lhd y \wedge \lambda(y) = j?i))$$

To ease the handling of $\mathrm{MSO}(\Sigma, C)$-formulas in the framework of $(\widetilde{\Sigma}, C)$-dags, we will henceforth use $x \lhd_i y$, for $i \in Ag$, to denote the formula $x \lhd y \wedge \lambda(x) \in \Sigma_i \wedge \lambda(y) \in \Sigma_i$ and we will use $x \lhd_{\mathrm{c}} y$ instead of $x \lhd y \wedge \bigvee_{(a,b)\in I_{\widetilde{\Sigma}}} \lambda(x) = a \wedge \lambda(y) = b$. To find suitable definitions of $\leq_i$ and $\lhd_{(i,j)}$, which shall correspond to the *relations* $\leq_i$ and, respectively, $\lhd_{(i,j)}$ as specified above, is left to the reader as an exercise.

Traces and MSCs have a nice property in common: any structure can be uniquely determined solely by the knowledge of its projections: given words $\underline{w}_i \in \mathbb{W}(\Sigma_i)$ for any $i \in Ag$, there is at most one trace $\mathcal{T} \in \mathbb{TR}^+(\widetilde{\Sigma})$ such that, for any $i \in Ag$, $\mathcal{T} \restriction i = \underline{w}_i$. In general, we will call a class $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$ *projective* if, for any collection $(\underline{w}_i)_{i\in Ag}$ of words $\underline{w}_i \in \mathbb{W}(\Sigma_i)$, there is at most one $(\widetilde{\Sigma}, C)$-dag $\mathcal{D} \in \mathcal{K}$ such that, for any $i \in Ag$, $\mathcal{D} \restriction i = \underline{w}_i$.

Observe that $\mathbb{DAG}(\widetilde{\Sigma}, C)$ and $\mathbb{LMSC}(Ag)$ are not projective, whereas $\mathbb{TR}^+(\widetilde{\Sigma})$, $\mathbb{TR}^-(\widetilde{\Sigma})$, and $\mathbb{MSC}(Ag)$ are. The notion of a projective class of $(\widetilde{\Sigma}, C)$-dags allows us to introduce *product languages*, which *shuffle* local projections in terms of word languages to obtain languages of dags. Those languages can usually be realized by simple and natural automata models (cf. Chaps. 6, 7, and 8).

5.2 The Operational Behavior of $(\widetilde{\Sigma}, C)$-Dags

A relation $\Longrightarrow_{(\widetilde{\Sigma},C)} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C) \times \mathbb{DAG}(\widetilde{\Sigma}, C)$ will reflect the operational behavior of $(\widetilde{\Sigma}, C)$-dags. As usual, we hereby write $\mathcal{D} \Longrightarrow_{(\widetilde{\Sigma},C)} \mathcal{D}'$ instead of $(\mathcal{D}, \mathcal{D}') \in \Longrightarrow_{(\widetilde{\Sigma},C)}$. Intuitively, $\mathcal{D}$ and $\mathcal{D}'$ are *configurations* of a system, and executing an action in configuration $\mathcal{D}$ may lead to configuration $\mathcal{D}'$. Given $(\widetilde{\Sigma}, C)$-dags $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda)$ and $\mathcal{D}' = (V', \{\lhd'_\ell\}_{\ell \in C}, \lambda')$, we have $\mathcal{D} \Longrightarrow_{(\widetilde{\Sigma},C)} \mathcal{D}'$ if $V' = V \uplus \{u\}$, $\lambda'_{|V} = \lambda$, and, for any $\ell \in C$, $\lhd'_\ell = \lhd_\ell \uplus \lhd''_\ell$ for some $\lhd''_\ell \subseteq V \times \{u\}$. This definition is canonically extended towards $\Longrightarrow_{(\widetilde{\Sigma},C),Q} \subseteq \langle \mathbb{DAG}(\widetilde{\Sigma}, C), Q\rangle \times \langle \mathbb{DAG}(\widetilde{\Sigma}, C), Q\rangle$ for some alphabet Q.

Example 5.13. Suppose $\widetilde{\Sigma} = (\{a\}, \{b\})$, $C = \{1, 2\}$, $Q = \{q_1, q_2\}$, and $\mathcal{D}_\mathrm{b}, \mathcal{D}_\mathrm{c}, \mathcal{D}_\mathrm{d} \in \langle \mathbb{DAG}(\widetilde{\Sigma}, C), Q\rangle$ are given by Fig. 5.6b–d, respectively. Then, we have both $\mathcal{D}_\mathrm{b} \Longrightarrow_{(\widetilde{\Sigma},C),Q} \mathcal{D}_\mathrm{c}$ and $\mathcal{D}_\mathrm{b} \Longrightarrow_{(\widetilde{\Sigma},C),Q} \mathcal{D}_\mathrm{d}$.

In the following, the elements of Q play the role of *states*. Thus, a configuration is supplemented by some additional information to make system behavior more flexible, in particular in terms of automata. The idea of automata running on $(\widetilde{\Sigma}, C)$-dags is to confine the behavior of $\Longrightarrow_{(\widetilde{\Sigma},C)}$ by means of *transitions*, i.e., moving from one configuration into another depends on the existence of a corresponding transition rule. Formally, the set of *transitions* over $(\widetilde{\Sigma}, C)$ and Q is given by $\mathit{Trans}_{(\widetilde{\Sigma},C)}(Q) := (Q \uplus \{-\})^{\Sigma \times C} \times \Sigma \times Q$. A triple $(\overline{q}, a, q) \in \mathit{Trans}_{(\widetilde{\Sigma},C)}(Q)$ might be read as follows: the system can execute an a and change into some local state q, if, for any $(b, \ell) \in \Sigma \times C$ with $\overline{q}[(b, \ell)] \neq -$, an action b, controlled by an ℓ-labeled edge, has led to state $\overline{q}[(b, \ell)]$. Given a system execution in terms of a Q-extended dag over $(\widetilde{\Sigma}, C)$, we can identify the transitions invoked by the system. So suppose we have $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, (\lambda, \rho)) \in \langle \mathbb{DAG}(\widetilde{\Sigma}, C), Q\rangle$. We define $\mathit{trans}_\mathcal{D} : V \to \mathit{Trans}_{(\widetilde{\Sigma},C)}(Q)$ and, for any $u \in V$, set $\mathit{trans}_\mathcal{D}(u)$ (the transition taken by the system when executing u) to be $(\overline{q}, \lambda(u), \rho(u))$ where, for any $(b, \ell) \in \Sigma \times C$,

$$\overline{q}[(b, \ell)] = \begin{cases} - & \text{if } (b, \ell) \notin \mathrm{Read}(u) \\ \rho((b, \ell)\text{-pred}(u)) & \text{if } (b, \ell) \in \mathrm{Read}(u) \end{cases}$$

In other words, the execution of u depends on its direct predecessor nodes. This view will be taken and be made more precise in the next section when we consider automata on $(\widetilde{\Sigma}, C)$-dags. A transition $t = (\overline{q}, a, q) \in \mathit{Trans}_{(\widetilde{\Sigma},C)}(Q)$ might be considered to be the Q-extended dag $\mathcal{D}(t) := (V \uplus \{q\}, \{\lhd_\ell\}_{\ell \in C}, \lambda)$ over (Σ, C) with

- $V = \{(b, \ell) \in \Sigma \times C \mid \overline{q}[(b, \ell)] \in Q\}$,
- $\lhd_\ell = \{(b, \ell') \in V \mid \ell' = \ell\} \times \{q\}$ for any $\ell \in C$,
- $\lambda((b, \ell)) = (b, \overline{q}[(b, \ell)])$ for any $(b, \ell) \in V$, and
- $\lambda(q) = (a, q)$.

Moreover, $\overline{q} \in (Q \uplus \{-\})^{\Sigma \times C}$ may be considered to be a subset of $(\Sigma \times Q) \times C$ with the understanding that, for any $(a, \ell) \in \Sigma \times C$ and $q \in Q$, $((a, q), \ell) \in \overline{q}$ iff $\overline{q}[(a, \ell)] = q$. In the following, we therefore often write a transition $(\overline{q}, a, q)$ as $\overline{q} \longrightarrow (a, q)$ with $\overline{q}$ being a subset of $(\Sigma \times Q) \times C$. Note that, if C is a singleton, we may consider a transition over $(\widetilde{\Sigma}, C)$ and Q to be an element of $(Q \uplus \{-\})^{\Sigma} \times \Sigma \times Q$

Let $\Delta \subseteq \mathit{Trans}_{(\widetilde{\Sigma}, C)}(Q)$. To indicate that a transition has been applied in accordance with the rules specified by Δ, we write, given $\mathcal{D}, \mathcal{D}' \in \langle \mathbb{DAG}(\widetilde{\Sigma}, C), Q \rangle$ and assuming that $(\widetilde{\Sigma}, C)$ will be clear from the context, $\mathcal{D} \Longrightarrow_{\Delta} \mathcal{D}'$ if both $\mathcal{D} \Longrightarrow_{(\widetilde{\Sigma}, C), Q} \mathcal{D}'$ and $\mathit{trans}_{\mathcal{D}'}(u) \in \Delta$ where u is the unique vertex of $\mathcal{D}'$ that does not occur in $\mathcal{D}$.

Example 5.14. Suppose $\widetilde{\Sigma} = (\{a\}, \{b\})$, $C = \{1, 2\}$, and $Q = \{q_1, q_2\}$. The graph $\mathcal{D}(t)$ of the transition $t = \{((a, q_1), 1), ((b, q_2), 2)\} \longrightarrow (b, q_2) \in \mathit{Trans}_{(\widetilde{\Sigma}, C)}(Q)$ is depicted in Fig. 5.6a. In view of Fig. 5.6b–d, $\mathcal{D}_{\mathrm{b}} \Longrightarrow_{\{t\}} \mathcal{D}_{\mathrm{c}}$ and $\mathcal{D}_{\mathrm{b}} \Longrightarrow_{\{t\}} \mathcal{D}_{\mathrm{d}}$.

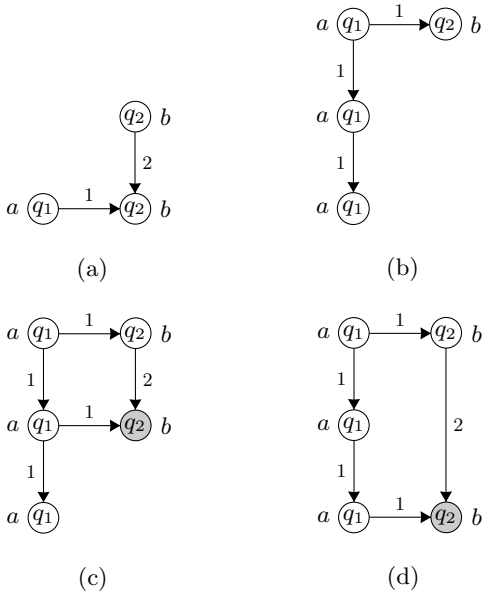


Fig. 5.6. The transitions of a system with $\widetilde{\Sigma} = (\{a\}, \{b\})$, $C = \{1, 2\}$, and $Q = \{q_1, q_2\}$

If we suppose a system execution to be completed, we might wish to identify the current state of any agent to determine if the execution is being accepted or not. If, however, an agent has not taken part in the system execution, then it might be in some initial state.

So suppose the finite set $\mathcal{O} \supseteq Q$ contains all the states from Q and all the possible initial states.[1] Suppose we are given $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, (\lambda, \rho)) \in \langle \mathbb{DAG}(\widetilde{\Sigma}, C), Q \rangle$ and suppose furthermore $\overline{q} \in \mathcal{O}^{Ag}$ is the *global initial state* of the system. We accordingly define $\mathit{final}^{\overline{q}}_{\mathcal{D}} \in \mathcal{O}^{Ag}$ and set $\mathit{final}^{\overline{q}}_{\mathcal{D}}[i]$ to be $\overline{q}[i]$ for any agent $i \in Ag$ with $V_i = \emptyset$. For any other agent i, we let $\mathit{final}^{\overline{q}}_{\mathcal{D}}[i] = \rho(u)$ where u is Σ_i-maximal in V. Thus, any agent contributes to $\mathit{final}^{\overline{q}}_{\mathcal{D}}$ the state of the vertex it has executed last (or $\overline{q}[i]$, the initial state of agent i, if there is no such maximal vertex). In other words, if the system starts in the global state $\overline{q}$ and executes $\mathcal{D}$, then it ends up in state $\mathit{final}^{\overline{q}}_{\mathcal{D}}$. Considering Fig. 5.6, then $\mathit{final}^{\overline{q}}_{\mathcal{D}_\mathrm{b}} = \mathit{final}^{\overline{q}}_{\mathcal{D}_\mathrm{c}} = \mathit{final}^{\overline{q}}_{\mathcal{D}_\mathrm{d}} = (q_1, q_2)$, no matter what the value of $\overline{q}$ is.

Some machines take their decision locally, i.e., an agent executes an event u depending on its current local state rather than on the collection of states assigned to u's direct predecessors. So again let $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, (\lambda, \rho)) \in \langle \mathbb{DAG}(\widetilde{\Sigma}, C), Q \rangle$ and suppose $\overline{q} \in \mathcal{O}^{Ag}$ and $* \notin \mathcal{O}$. Applied to some agent $i \in Ag$, the tuple $\mathit{source}^{\overline{q}}_{\mathcal{D}}(u) \in (\mathcal{O} \cup \{*\})^{Ag}$ with $u \in V$ yields the current state of i before executing u. Provided $u \in V_i$ and u is not the first event executed by i, we deal with the state assigned to the event that i has executed right before u. In case that u is the first event executed by i, however, we deal with the initial state $\overline{q}[i]$. If, otherwise, $u \notin V_i$, then $\mathit{source}^{\overline{q}}_{\mathcal{D}}(u)[i]$ is set to be $*$, which is a kind of "don't care" or "don't know" statement reflecting that agent j with $u \in V_j$ might not be aware of how far agent i has proceeded. More precisely, for $i \in Ag$, $\mathit{source}^{\overline{q}}_{\mathcal{D}}(u)[i]$ is given as follows:

$$\mathit{source}^{\overline{q}}_{\mathcal{D}}(u)[i] = \begin{cases} \overline{q}[i] & \text{if } u \in V_i \text{ and } u = \mathit{first}(\mathcal{D} \restriction i) \\ \rho(\mathit{last}((\mathcal{D} \restriction i) \downarrow u)) & \text{if } u \in V_i \text{ and } u \neq \mathit{first}(\mathcal{D} \restriction i) \\ * & \text{if } u \notin V_i \end{cases}$$

5.3 Asynchronous Cellular Automata with Types

We now propose asynchronous cellular automata with types (ACATs) running on $(\widetilde{\Sigma}, C)$-dags, which combine the models of asynchronous cellular automata and graph acceptors and allow a uniform embedding of many existing models of concurrency. In particular, an ACAT has (limited) access to the future to articulate communication requests. A communication request is reflected by a type function, which associates with any action a and any state q the set of actions that henceforth "communicate" with a, provided a has been executed in state q. Regarding LMSCs, for example, we might require an event labeled with a send action 1!2 to be followed by the suitable receive event, which is then labeled with 2?1. ACATs turn out to have the same expressive power as EMSO logic relative to any class of $(\widetilde{\Sigma}, C)$-dags.

[1] In our model of an asynchronous cellular automaton, initial states are actually not part of the set of states, which makes their notation slightly easier. For the moment, however, the reader may assume $\mathcal{O} = Q$ as well.

Definition 5.15 (Asynchronous Cellular Automaton with Types). *An* asynchronous cellular automaton with types *(ACAT) over* $(\widetilde{\Sigma}, C)$ *is a structure* (Q, Δ, T, F) *where*

- Q *is the nonempty finite set of* states,
- $\Delta \subseteq \mathit{Trans}_{(\widetilde{\Sigma},C)}(Q)$ *is the set of* transitions,
- $T : (\Sigma \times Q) \to 2^{\Sigma \times C}$ *is the* type function, *and*
- $F \subseteq (Q \uplus \{\imath\})^{Ag}$ *is the set of* final states.

So let $\mathcal{A} = (Q, \Delta, T, F)$ be an ACAT over $(\widetilde{\Sigma}, C)$ and, moreover, suppose $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda)$ to be a $(\widetilde{\Sigma}, C)$-dag. A run of $\mathcal{A}$ on $\mathcal{D}$ is a mapping $\rho : V \to Q$ such that, for any $u \in V$, $\mathit{trans}_{(\mathcal{D},\rho)}(u) \in \Delta$. Moreover, ρ is *accepting* if both $\mathit{final}_{(\mathcal{D},\rho)}^{(\imath)_{i \in Ag}} \in F$ and, for any $u \in V$, we have $T(\lambda(u), \rho(u)) \subseteq \mathrm{Write}(u)$. The intuition behind the latter condition is that we require $\mathrm{Write}(u)$ to contain at least the communication requests imposed by the type function of the automaton.[2] If C is a singleton, T might be seen as a mapping $(\Sigma \times Q) \to 2^{\Sigma}$. Given $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$, we denote by $L_{\mathcal{K}}(\mathcal{A})$ the *language* of $\mathcal{A}$ relative to $\mathcal{K}$, i.e., the set of $(\widetilde{\Sigma}, C)$-dags $\mathcal{D} \in \mathcal{K}$ such that there is an accepting run of $\mathcal{A}$ on $\mathcal{D}$. However, we usually write $L(\mathcal{A})$ instead of $L_{\mathbb{DAG}(\widetilde{\Sigma},C)}(\mathcal{A})$.

The following lemma justifies ACATs being classified as an operational automata model.

Lemma 5.16. *Let* $\mathcal{A} = (Q, \Delta, T, F)$ *be an ACAT over* $(\widetilde{\Sigma}, C)$ *and, moreover, let* $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda)$ *be a* $(\widetilde{\Sigma}, C)$*-dag. A mapping* $\rho : V \to Q$ *is a run of* $\mathcal{A}$ *on* $\mathcal{D}$ *iff* $(\emptyset, \{\emptyset\}_{\ell \in C}, \emptyset) \Longrightarrow_{\Delta}^{*} (\mathcal{D}, \rho)$.

Exercise 5.17. Prove Lemma 5.16.

Definition 5.18. *An ACAT* (Q, Δ, T, F) *over* $(\widetilde{\Sigma}, C)$ *is an* asynchronous cellular automaton *(ACA) over* $(\widetilde{\Sigma}, C)$ *if* $T(a, q) = \emptyset$ *for any* $a \in \Sigma$ *and* $q \in Q$.

Given $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$, let $\mathcal{ACAT}(\widetilde{\Sigma}, C)_{\mathcal{K}} := \{L \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C) \mid L = L_{\mathcal{K}}(\mathcal{A})$ for some ACAT $\mathcal{A}$ over $(\widetilde{\Sigma}, C)\}$. The class $\mathcal{ACA}(\widetilde{\Sigma}, C)_{\mathcal{K}}$ is defined accordingly. Also, $\mathcal{ACAT}(\widetilde{\Sigma}, C)$ and $\mathcal{ACA}(\widetilde{\Sigma}, C)$ shall abbreviate the classes $\mathcal{ACAT}(\widetilde{\Sigma}, C)_{\mathbb{DAG}(\widetilde{\Sigma},C)}$ and $\mathcal{ACA}(\widetilde{\Sigma}, C)_{\mathbb{DAG}(\widetilde{\Sigma},C)}$, respectively. As usual, if C is negligible, then we simply refer to $\widetilde{\Sigma}$, writing, for example, $\mathcal{ACA}(\widetilde{\Sigma})_{\mathbb{DAG}(\widetilde{\Sigma})}$.

Let $\mathcal{A} = (Q, \Delta, T, F)$ be an ACAT over $(\widetilde{\Sigma}, C)$. We call $\mathcal{A}$ *deterministic* if, for any $\overline{q} \in (Q \uplus \{-\})^{\Sigma \times C}$ and $a \in \Sigma$, there is at most one state $q \in Q$ such that $\overline{q} \longrightarrow (a, q) \in \Delta$. The corresponding language classes are denoted by det-$\mathcal{ACAT}(\widetilde{\Sigma}, C)_{\mathcal{K}}$. For $a \in \Sigma$, let Q_a denote the set $\{q \in Q \mid$ there is $(\overline{q}, b, q') \in \Delta$ such that $(b = a$ and $q' = q)$ or $\overline{q}[(a, \ell)] = q$ for some $\ell \in C\}$. We call $\mathcal{A}$ *state separated* if the sets Q_a, $a \in \Sigma$, are disjoint.

[2] Note that we could even require $T(\lambda(u), \rho(u)) = \mathrm{Write}(u)$ without affecting the expressiveness of ACATs.

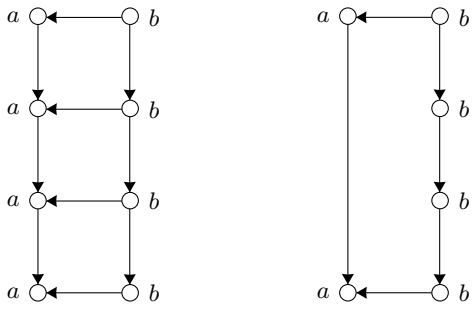


Fig. 5.7. The $(\{a\},\{b\})$-dags $\mathcal{D}_4$ and $\mathcal{D}_4[1,3]$ for Example 5.19 and the proof of Lemma 5.27, respectively

Finally, given a class $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$, $\mathcal{A}$ is called *adjusted* to $\mathcal{K}$ if, for any $t \in \Delta$, there are a Q-extended $(\widetilde{\Sigma}, C)$-dag $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \langle \mathcal{K}, Q \rangle$ and a node $u \in V$ such that $trans_{\mathcal{D}}(u) = t$. Obviously, we can assume $\mathcal{A}$ to be adjusted to $\mathcal{K}$ if we consider $\mathcal{A}$ relative to that class.

Example 5.19. Suppose $\widetilde{\Sigma} = (\{a\},\{b\})$ and let L be the set of $\widetilde{\Sigma}$-dags $\mathcal{D}_n = (V_n, \lhd_n, \lambda_n)$, $n \geq 1$, where $V_n = \{u_1, \ldots, u_n, v_1, \ldots, v_n\}$, $\lhd_n$ is the union of $\{(u_i, u_{i+1}) \mid i \in \{1, \ldots, n-1\}\}$, $\{(v_i, v_{i+1}) \mid i \in \{1, \ldots, n-1\}\}$, and $\{(v_i, u_i) \mid i \in \{1, \ldots, n\}\}$, and the u_i are labeled by λ_n with a, whereas the v_i are labeled with b. For example, Fig. 5.7a illustrates $\mathcal{D}_4$. An ACAT $\mathcal{A}$ over $\widetilde{\Sigma}$ with $L(\mathcal{A}) = L$ is (Q, Δ, T, F) where $Q = \{q_1, q_2\}$ and Δ contains the following transitions (recall that there are no edge labelings):

$$\begin{aligned} \emptyset &\longrightarrow (b, q_2) \\ \{(b, q_2)\} &\longrightarrow (b, q_2) \\ \{(b, q_2)\} &\longrightarrow (a, q_1) \\ \{(a, q_1), (b, q_2)\} &\longrightarrow (a, q_1) \end{aligned}$$

Moreover, we set

$$\begin{aligned} T(a, q_1) &= \emptyset \\ T(b, q_1) &= \emptyset \\ T(a, q_2) &= \emptyset \\ T(b, q_2) &= \{a\} \end{aligned}$$

and $F = \{(q_1, q_2)\}$. The transitions and the type function of $\mathcal{A}$ can be depicted as shown in Fig. 5.8, i.e., solid arrows and the associated nodes provide graph patterns in terms of labelings and states, while the target of a dashed arrow indicates a communication request. An accepting run of $\mathcal{A}$ on $\mathcal{D}_n$ assigns q_1 to any a-labeled vertex and q_2 to any b-labeled one. Observe that $\mathcal{A}$ is deterministic and state separated. We will later see that $L \notin \mathcal{ACA}(\widetilde{\Sigma})$.

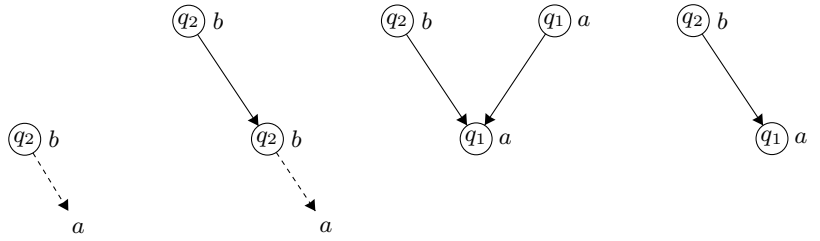


Fig. 5.8. Transitions and type function of an ACAT

Example 5.20. Again, let $\widetilde{\Sigma} = (\{a\}, \{b\})$ and let L be the set of $\widetilde{\Sigma}$-dags $\mathcal{D}_n = (V_n, \lhd_n, \lambda_n)$, $n \geq 1$, where, again, $V_n = \{u_1, \ldots, u_n, v_1, \ldots, v_n\}$, $\lhd_n$ is the union of $\{(u_i, u_{i+1}) \mid i \in \{1, \ldots, n-1\}\}$, $\{(v_i, v_{i+1}) \mid i \in \{1, \ldots, n-1\}\}$, $\{(v_i, u_i) \mid i \in \{1, \ldots, n\}\}$, and $\{(u_i, v_{i+1}) \mid i \in \{1, \ldots, n-1\}\}$. Moreover, λ_n labels u_i with a, whereas the v_i are labeled with b. $\mathcal{D}_4$ is depicted in Fig. 5.9. It turns out that, this time, even an ACA is sufficient to recognize L relative to $\mathbb{DAG}(\widetilde{\Sigma})$. Namely, an ACA $\mathcal{A}$ over $\widetilde{\Sigma}$ with $L(\mathcal{A}) = L$ is given by (Q, Δ, T, F) specified as follows. The set of states is $Q = \{\mathit{true}, \mathit{false}\}$. If a state is assigned to an a-labeled node, it indicates whether it is covered by a b-labeled node ($=\mathit{false}$) or not ($=\mathit{true}$). Obviously, the corresponding process must exhibit *true* in a final state. If a state is assigned to a b-labeled node, it indicates whether we deal with the first b-labeled node ($=\mathit{true}$) or not ($=\mathit{false}$). This is to guarantee that initial transitions can be applied only once to guarantee the required edge structure of a dag. These ideas are reflected by Δ, which contains the following transitions:

$$\begin{aligned}
\emptyset &\longrightarrow (b, \mathit{true}) \\
\{(b, \mathit{true})\} &\longrightarrow (a, q) \quad \text{for any } q \in Q \\
\{(a, \mathit{false}), (b, q)\} &\longrightarrow (b, \mathit{false}) \text{ for any } q \in Q \\
\{(a, q_1), (b, q_2)\} &\longrightarrow (a, q) \quad \text{for any } q, q_1, q_2 \in Q
\end{aligned}$$

Moreover, we set $F = \{(\mathit{true}, \mathit{true}), (\mathit{true}, \mathit{false})\}$. Though $\mathcal{A}$ is adjusted to $\mathbb{DAG}(\widetilde{\Sigma})$, it is neither deterministic nor state separated.

Exercise 5.21. Specify ACATs over $\widetilde{\Sigma} = (\{a, b\}, \{b, c\})$ that recognize, relative to $\mathbb{DAG}(\widetilde{\Sigma})$, the sets of $\widetilde{\Sigma}$-dags $(V, \lhd, \lambda)$ such that

(a) for any $u \in V_a$, we have $b \in \mathrm{Write}(u)$ or $c \in \mathrm{Write}(u)$,
(b) there are nodes $u \in V_a$ and $v \in V_c$ satisfying neither $u \leq v$ nor $v \leq u$,
(c) there are nodes $u \in V_a$ and $v \in V_c$ satisfying $u \leq v$.

Exercise 5.22. Show that, for any deterministic ACAT $\mathcal{A}$ over $(\widetilde{\Sigma}, C)$, there is an ACAT $\mathcal{A}'$ over $(\widetilde{\Sigma}, C)$ such that both $L(\mathcal{A}') = L(\mathcal{A})$ and, for any $\mathcal{D} \in \mathbb{DAG}(\widetilde{\Sigma}, C)$, there is exactly one run of $\mathcal{A}'$ on $\mathcal{D}$.

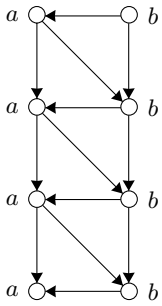


Fig. 5.9. $\mathcal{D}_4$ for Example 5.20

Exercise 5.23. Show that any ACAT $\mathcal{A}$ over $(\widetilde{\Sigma}, C)$ can be transformed into a state separated ACAT $\mathcal{A}'$ over $(\widetilde{\Sigma}, C)$ such that $L(\mathcal{A}') = L(\mathcal{A})$.

Exercise 5.24. Prove that the language $L_{\mathbb{DAG}_{\rightrightarrows}(\widetilde{\Sigma})}(\mathcal{A})$ of an ACAT $\mathcal{A} = (Q, \Delta, T, F)$ over $\widetilde{\Sigma}$ remains the same if we remove from Δ any transition $\overline{q} \longrightarrow (a, q)$ such that

(a) $\overline{q}[b] \neq -$ and $\overline{q}[c] \neq -$ for some distinct actions b and c with $b \; D_{\widetilde{\Sigma}} \; c$,
(b) $T(a, q)$ contains distinct actions b and c with $b \; D_{\widetilde{\Sigma}} \; c$.

Exercise 5.25. Show that, without loss of generality, we may assume an ACAT $\mathcal{A} = (Q, \Delta, T, F)$ over $(\widetilde{\Sigma}, C)$ to possess the following property: for any $\overline{q} \in F$, any $i \in Ag$ with $\overline{q}[i] \in Q$, any $(\widetilde{\Sigma}, C)$-dag $\mathcal{D} = (V, \lhd, \lambda)$, any $u \in V$, and any run ρ of $\mathcal{A}$ on $\mathcal{D}$, if $\rho(u) = \overline{q}[i]$, then u is Σ_i-maximal.

Exercise 5.26. Show that the expressiveness of ACATs does not change if, in a run ρ, we require $T(\lambda(u), \rho(u)) = \mathrm{Write}(u)$ for any u instead of $T(\lambda(u), \rho(u)) \subseteq \mathrm{Write}(u)$.

5.4 ACATs vs. ACAs

In what follows, we compare in detail the expressiveness of ACATs and ACAs. Note that some results rely on special (though simple) structures of the underlying distributed alphabet.

Lemma 5.27 (cf. [29]). *In general,*

$$\mathcal{ACA}(\widetilde{\Sigma}) \subsetneqq \mathcal{ACAT}(\widetilde{\Sigma})$$

Proof. Set $\widetilde{\Sigma} = (\{a\}, \{b\})$. The language L from Example 5.19 cannot be recognized by some ACA over $\widetilde{\Sigma}$ relative to $\mathbb{DAG}(\widetilde{\Sigma})$. For suppose there is an ACA $\mathcal{A}$ over $\widetilde{\Sigma}$ with $L_{\mathbb{DAG}(\widetilde{\Sigma})}(\mathcal{A}) = L$. For $n \geq 1$, suppose furthermore ρ to be a run of $\mathcal{A}$ on $\mathcal{D}_n$. If n has been chosen large enough, there are $1 \leq i < j < n$ such that $\rho(u_i) = \rho(u_j)$. From $\mathcal{D}_n$, we obtain the $\widetilde{\Sigma}$-dag $\mathcal{D}_n[i,j]$ by removing from V_n the vertices $u_{i+1}, \ldots, u_j$ and from $\lhd_n$ any edge touching some of these nodes and by adding instead an edge from u_i to u_{j+1}. Though $\mathcal{D}_n[i,j] \notin L$, $\mathcal{A}$ admits an accepting run on $\mathcal{D}_n[i,j]$. The reason is that, in the example, $\mathcal{A}$ has no means to impose on a b-labeled vertex an a-labeled successor. □

However, the language L from the above proof is obviously $\mathrm{FO}_{\mathbb{DAG}(\widetilde{\Sigma})}$-definable. Moreover, it is easy to provide a language from $\mathcal{ACA}(\widetilde{\Sigma})$ that is not $\mathrm{FO}_{\mathbb{DAG}(\widetilde{\Sigma})}$-definable. This leads to the following corollary:

Corollary 5.28. *The classes $\mathcal{FO}_{\mathbb{DAG}(\widetilde{\Sigma})}$ and $\mathcal{ACA}(\widetilde{\Sigma})$ are incomparable with respect to inclusion.*

Though ACATs are generally strictly more expressive than ACAs, many distributed systems allow for dropping types. For example, if we know from the labeling of a node whether it has a communication partner in terms of an independent event or not, this allows us to reduce the type function accordingly.

Definition 5.29 (Communication Closed). *A class $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$ is called* communication closed *if, for any $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda), (V', \{\lhd'_\ell\}_{\ell \in C}, \lambda') \in \mathcal{K}$, any $a \in \Sigma$, and any $u \in V$ and $u' \in V'$ with $\lambda(u) = \lambda'(u') = a$, we have $\{(b,\ell) \in \mathrm{Write}(u) \mid b\ I_{\widetilde{\Sigma}}\ a\} = \{(b,\ell) \in \mathrm{Write}(u') \mid b\ I_{\widetilde{\Sigma}}\ a\}$.*

If, in addition, a local process proceeds in a sequential fashion to access its last state when executing a further action, we can even do without type functions.

Definition 5.30 (Locally Covering). *A class $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$ is called* locally covering *if both*

- *for any $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathcal{K}$, $\{(u,v) \in \lhd \mid \lambda(u)\ D_{\widetilde{\Sigma}}\ \lambda(v)\} = \bigcup_{i \in Ag} \lessdot_i$ (recall that $\lessdot_i$ is the cover relation of the total order $\leq_i$), and*
- *there is a mapping $\chi : (\Sigma \times C \times \Sigma) \to 2^{Ag}$ such that, for any $(\widetilde{\Sigma}, C)$-dag $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathcal{K}$, any $\ell \in C$, and any $(u,v) \in \lhd_\ell$, $\chi(\lambda(u), \ell, \lambda(v)) = \{i \in Ag \mid u \lessdot_i v\}$.*

Thus, a class that is locally covering allows us to infer, from the labeling of an edge and the labelings of the associated nodes u and v (say, with $u \lhd v$), the set of agents that execute v directly after u.

Lemma 5.31. *Let $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$. If $\mathcal{K}$ is both communication closed and locally covering, then*

$$\mathcal{ACA}(\widetilde{\Sigma}, C)_{\mathcal{K}} \ = \ \mathcal{ACAT}(\widetilde{\Sigma}, C)_{\mathcal{K}}$$

Proof. Let $\mathcal{A} = (Q, \Delta, T, F)$ be an ACAT over $(\widetilde{\Sigma}, C)$. Moreover, suppose $\mathcal{K} \subseteq \mathbb{DAG}(\widetilde{\Sigma}, C)$ is communication closed and locally covering where χ shall be the required mapping $(\Sigma \times C \times \Sigma) \to 2^{Ag}$. Without loss of generality, we furthermore assume that $\mathcal{A}$ is state separated. To construct an ACA that is equivalent to $\mathcal{A}$ relative to $\mathcal{K}$, we perform the following steps.

1. For any $(a, q) \in \Sigma \times Q$ and $(b, \ell) \in T(a, q)$: if, for any $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathcal{K}$ and $u \in V_a$, we have $(b, \ell) \notin \mathrm{Write}(u)$, then
   - remove from Δ any transition $\overline{q} \longrightarrow (a, q)$ and
   - set $T(a, q) = \emptyset$.
2. For any $a, b \in \Sigma$ with $a \; I_{\widetilde{\Sigma}} \; b$ and any $\ell \in C$ such that $\mathcal{K}$ (which is communication closed) requires an a-labeled vertex u to be directly followed by some b-labeled vertex v satisfying $u \lhd_\ell v$, we can, without changing the recognized language relative to $\mathcal{K}$,
   - remove (b, ℓ) from any communication request $T(a, q)$.
3. For any $(a, q) \in \Sigma \times Q$ and any remaining $(b, \ell) \in T(a, q)$ (note that, henceforth, we have both $a \; D_{\widetilde{\Sigma}} \; b$ and $\chi(a, \ell, b) \neq \emptyset$), we may
   - remove (b, ℓ) from $T(a, q)$,
   - remove from Δ any transition $\overline{q} \longrightarrow (c, q')$ such that there is $\ell' \in C$ satisfying $\overline{q}[(a, \ell')] = q$, $(c, \ell') \neq (b, \ell)$, and $\chi(a, \ell, b) \cap \chi(a, \ell', c) \neq \emptyset$, and
   - remove from F any $\overline{q} \in (Q \cup \{\imath\})^{Ag}$ with $\overline{q}[i] = q$ for some $i \in \chi(a, \ell, b)$.

In fact, we finally gain an ACA (thus, with negligible type function), which, moreover, is equivalent to $\mathcal{A}$ relative to $\mathcal{K}$. □

The classes $\mathbb{TR}^+(\widetilde{\Sigma})$ and $\mathbb{TR}^-(\widetilde{\Sigma})$ of M^+- and, respectively, M^--traces are trivially communication closed, as no pair of neighboring nodes can be labeled with independent actions. However, $\mathbb{TR}^+(\widetilde{\Sigma})$ is locally covering, whereas $\mathbb{TR}^-(\widetilde{\Sigma})$ is not. The required mapping for $\mathbb{TR}^+(\widetilde{\Sigma})$ simply assigns ℓ to any triple $(a, \ell, b) \in \Sigma \times 2^{Ag} \times \Sigma$. In contrast, an event in a trace from $\mathbb{TR}^-(\widetilde{\Sigma})$ can access only the nodes it *covers* with respect to $\lessdot$. The class $\mathbb{MSC}(Ag)$ is communication closed, too: any sending vertex has exactly one successor vertex, labeled with the corresponding receive action, that is not controlled by the same agent. However, the class $\mathbb{LMSC}(Ag)$ of lossy MSCs is not communication closed and can no longer do without types, as an easy adaption of the proof of Lemma 5.27 shows. The reader may verify that, however, both $\mathbb{MSC}(Ag)$ and $\mathbb{LMSC}(Ag)$ are locally covering.

Corollary 5.32.

(a) $\mathcal{ACA}(\widetilde{\Sigma}, 2^{Ag})_{\mathbb{TR}^+(\widetilde{\Sigma})} \;=\; \mathcal{ACAT}(\widetilde{\Sigma}, 2^{Ag})_{\mathbb{TR}^+(\widetilde{\Sigma})}$,
(b) $\mathcal{ACA}(\widetilde{\Gamma}(Ag))_{\mathbb{MSC}(Ag)} \;=\; \mathcal{ACAT}(\widetilde{\Gamma}(Ag))_{\mathbb{MSC}(Ag)}$.

Note that, however, we will argue that $\mathcal{ACA}(\widetilde{\Sigma})_{\mathbb{TR}^-(\widetilde{\Sigma})} \;=\; \mathcal{ACAT}(\widetilde{\Sigma})_{\mathbb{TR}^-(\widetilde{\Sigma})}$ holds as well (cf. Chap. 6).

Exercise 5.33. Suppose $\widetilde{\Sigma} = (\{a, b\}, \{b, c\})$. Are the collections of $\widetilde{\Sigma}$-dags $(V, \lhd, \lambda)$ such that

(a) $V_b = \emptyset$,
(b) $V_c = \emptyset$,
(c) V is totally ordered by $\leq$,
(d) for any $(u, v) \in \lhd$, $u \in V_a$ implies $v \in V_c$ and $u \in V_c$ implies $v \in V_b$,

communication closed or not? Which are locally covering?

Lemma 5.34. *In general,*

$$\mathbb{DAG}_{\rightrightarrows}(\widetilde{\Sigma}) \;\not\in\; \mathcal{ACA}(\widetilde{\Sigma})$$

Proof. Let us fix the distributed alphabet $\widetilde{\Sigma} = (\{a\}, \{c, d\})$. We proceed similarly to the proof of Lemma 5.27 and consider fifo-dags over $\widetilde{\Sigma}$ of the form presented in Fig. 5.10a, which are basically equipped with two rows of nodes, the one labeled with a and the other labeled with either c or d. Clearly, there must be such a fifo-dag $\mathcal{D} = (V, \lhd, \lambda)$ and an accepting run ρ of the ACA $\mathcal{A}$ at hand on $\mathcal{D}$ such that the pattern illustrated in the figure (i.e., with nodes labeled a, c, and d and states, say, q_a, q_c, and q_d) occurs twice. Merging the two c-labeled nodes and both d-labeled nodes while removing any other node in that row in between the two patterns results in a $\widetilde{\Sigma}$-dag such as the one illustrated by Fig. 5.10b, which will be accepted by $\mathcal{A}$ but is not a fifo-dag over $\widetilde{\Sigma}$. □

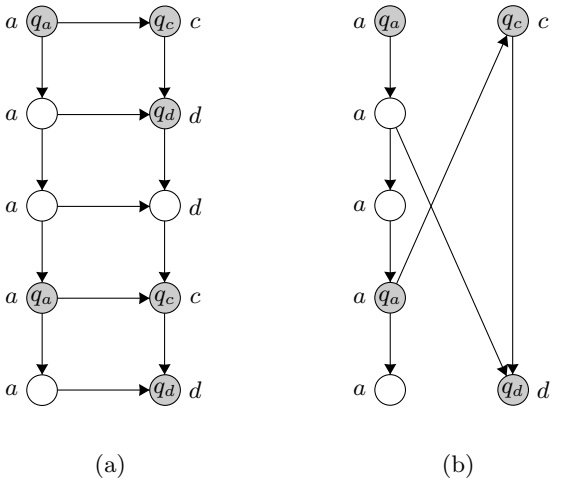


Fig. 5.10. A fifo-dag over $(\{a\}, \{c, d\})$ and an $(\{a\}, \{c, d\})$-dag that is not a fifo-dag over $(\{a\}, \{c, d\})$

Exercise 5.35. Show or disprove:

(a) $\mathbb{DAG}_{\rightrightarrows}((\{a,b\},\{b,c\})) \in \mathcal{ACA}((\{a,b\},\{b,c\}))$,
(b) $\mathbb{DAG}_{\rightrightarrows}((\{a,b\},\{b,c\})) \in \mathcal{ACA}((\{a\},\{b\},\{c\}))$.

Exercise 5.36. Show that

(a) $\mathbb{MSC} \notin \mathcal{ACA}(\widetilde{\Gamma})$,
(b) $\mathbb{MSC} \in \mathcal{ACAT}(\widetilde{\Gamma})$,
(c) $\mathbb{LMSC} \in \mathcal{ACA}(\widetilde{\Gamma})$.

Given two ACATs, one can obviously construct ACATs to recognize their intersection and union.

Lemma 5.37. *$\mathcal{ACAT}(\widetilde{\Sigma}, C)$ is closed under union and intersection.*

Proof. Suppose $\mathcal{A}_1 = (Q_1, \Delta_1, T_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Delta_2, T_2, F_2)$ are ACATs over $(\widetilde{\Sigma}, C)$. We build $\mathcal{A}_1 \times \mathcal{A}_2 = (Q, \Delta, T, F)$, an ACAT over $(\widetilde{\Sigma}, C)$ such that $L(\mathcal{A}_1 \times \mathcal{A}_2) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. It is given by

- $Q = Q_1 \times Q_2$,
- $\Delta = \{(\overline{q}, b, q) \in \mathit{Trans}_{(\widetilde{\Sigma},C)}(Q) \mid$ there are $(\overline{q}_1, b, q_1) \in \Delta_1$ and $(\overline{q}_2, b, q_2) \in \Delta_2$ such that, for any $(a,\ell) \in \Sigma \times C$, both $\overline{q}[(a,\ell)] = -$ implies $\overline{q}_1[(a,\ell)] = \overline{q}_2[(a,\ell)] = -$ and $\overline{q}[(a,\ell)] \in Q$ implies $\overline{q}[(a,\ell)] = (\overline{q}_1[(a,\ell)], \overline{q}_2[(a,\ell)])\}$,
- $T(a,(q_1,q_2)) = T_1(a,q_1) \cup T_2(a,q_2)$ for any $a \in \Sigma$ and $(q_1,q_2) \in Q$, and
- $F = \{\overline{q} \in (Q \cup \{\imath\})^{Ag} \mid$ there are $\overline{q}_1 \in F_1$ and $\overline{q}_2 \in F_2$ such that, for any $i \in Ag$, both $\overline{q}[i] = \imath$ implies $\overline{q}_1[i] = \overline{q}_2[i] = \imath$ and $\overline{q}[i] \in Q$ implies $\overline{q}[i] = (\overline{q}_1[i], \overline{q}_2[i])\}$.

The proof of closure under union proceeds similarly. Basically, the new state space is the disjoint union of the original ones. Accordingly, the same applies to the transitions. Proof details are left to the reader as an exercise. □

5.5 The Expressive Equivalence of ACATs and EMSO Logic

In this section, we establish the expressive equivalence of ACATs and EMSO logic. The easier part is to provide an EMSO formula for a given ACAT. We hereby mainly follow similar constructions applied, for example, to finite and the similar asynchronous automata (cf. Theorem 4.10 and [29] for examples).

Lemma 5.38.

$$\mathcal{ACAT}(\widetilde{\Sigma}, C) \;\subseteq\; \mathcal{EMSO}(\Sigma, C)_{\mathbb{DAG}(\widetilde{\Sigma},C)}$$

Proof. The construction of an EMSO sentence from a given ACAT proceeds as follows. Basically, an interpretation of second-order variables (actually, an interpretation that stands for a partition $(X_q)_{q\in Q}$ of the set of vertices at hand) means an assignment of states to vertices, which is then checked within the first-order fragment of the formula for being an accepting run. For simplicity, we suppose $Ag = \{1,\dots,K\}$ for some $K \geq 2$. Moreover, suppose $\mathcal{A} = (Q, \Delta, T, F)$ to be an ACAT over $(\widetilde{\Sigma}, C)$ with set of states $Q = \{1,\dots,n\}$. To check if, at the end of a run on a dag, the system is in some global final state $\overline{q} \in F$, we assume that, for any agent $i \in Ag$ with $\overline{q}[i] \neq \imath$, there is a set of nodes X that is upwards-closed and contains exactly one node x belonging to i. In that case, x is Σ_i maximal and shall be contained in $X_{\overline{q}[i]}$. Therefore, we define a formula $\max_i(x, X)$, which is satisfied if both X is upward-closed and x is the only node contained in X that is labeled with some $a \in \Sigma_i$:

$$\begin{aligned}\max_i(x, X) := \quad & \lambda(x) \in \Sigma_i \\ & \wedge\, x \in X \\ & \wedge\, \forall y \forall z\, ((y \in X \wedge y \lhd z) \rightarrow z \in X) \\ & \wedge\, \forall y\, ((y \in X \wedge \lambda(y) \in \Sigma_i) \rightarrow y = x)\end{aligned}$$

To ensure that the past of a node x corresponds to a transition, we use

$$\begin{aligned}&\mathrm{Trans}(x, X_1, \dots, X_n) := \\ &\qquad \bigvee_{\{((a_1,q_1),\ell_1),\dots,((a_m,q_m),\ell_m)\} \longrightarrow (a,q) \in \Delta} \\ &\qquad\quad x \in X_q \\ &\qquad \wedge\, \lambda(x) = a \\ &\qquad \wedge\, \textstyle\bigwedge_{k\in\{1,\dots,m\}} \exists y (y \lhd_{\ell_k} x \wedge \lambda(y) = a_k \wedge y \in X_k) \\ &\qquad \wedge\, \forall y \left(y \lhd x \rightarrow \textstyle\bigvee_{k\in\{1,\dots,m\}} (y \lhd_{\ell_k} x \wedge \lambda(y) = a_k) \right)\end{aligned}$$

To guarantee acceptance, we require formulas

$$\begin{aligned}&\mathrm{Type}(X_1, \dots, X_n) := \\ &\forall x \ \textstyle\bigwedge_{a\in\Sigma,\ q\in Q} \left((\lambda(x) = a \wedge x \in X_q) \rightarrow \bigwedge_{(b,\ell)\in T(a,q)} \exists y (x \lhd_\ell y \wedge \lambda(y) = b) \right)\end{aligned}$$

simulating T and, for any $\overline{q} \in (Q \cup \{\imath\})^{Ag}$,

$$\begin{aligned}\mathrm{Final}_{\overline{q}}(X_1, \dots, X_n, Y_1, \dots, Y_K) := \\ \textstyle\bigwedge_{i\in Ag,\ \overline{q}[i]\in Q} \exists x \left(\max_i(x, Y_i) \wedge x \in X_{\overline{q}[i]} \right) \\ \wedge \textstyle\bigwedge_{i\in Ag,\ \overline{q}[i]=\imath} \neg\exists x\ \lambda(x) \in \Sigma_i\end{aligned}$$

The above formulas can now be combined towards an EMSO(Σ, C)-sentence Ψ with $L_{\mathbb{DAG}(\widetilde{\Sigma},C)}(\Psi) = L(\mathcal{A})$ as follows:

$$\begin{aligned}\Psi = \exists X_1 \ldots \exists X_n\ \exists Y_1 \ldots \exists Y_K & \\ & \text{partition}(X_1, \ldots, X_n) \\ \wedge\ & \forall x\ \text{Trans}(x, X_1, \ldots, X_n) \\ \wedge\ & \text{Type}(X_1, \ldots, X_n) \\ \wedge\ & \bigvee_{\overline{q} \in F} \text{Final}_{\overline{q}}(X_1, \ldots, X_n, Y_1, \ldots, Y_K)\end{aligned}$$

Recall that $\text{partition}(X_1, \ldots, X_n)$, $\text{Trans}(x, X_1, \ldots, X_n)$, $\text{Type}(X_1, \ldots, X_n)$, and $\text{Final}_{\overline{q}}(X_1, \ldots, X_n, Y_1, \ldots, Y_K)$ are first-order formulas without second-order quantifiers. □

The following proposition is one step towards the transformation of an EMSO sentence into an equivalent ACAT.

Proposition 5.39. *Let $R \in \mathbb{N}$. There are an ACAT $\mathcal{A}_R = (Q, \Delta, T, F)$ over $(\widetilde{\Sigma}, C)$ and a mapping $\eta : Q \to R\text{-Sph}(\mathbb{DAG}(\widetilde{\Sigma}, C))$ such that $L(\mathcal{A}_R) = \mathbb{DAG}(\widetilde{\Sigma}, C)$ and, for any $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda) \in \mathbb{DAG}(\widetilde{\Sigma}, C)$, any accepting run ρ of $\mathcal{A}_R$ on $\mathcal{D}$, and any $u \in V$, we have $\eta(\rho(u)) = R\text{-Sph}(\mathcal{D}, u)$.*

Proof. Set $\mathcal{S}$ to be $R\text{-Sph}(\mathbb{DAG}(\widetilde{\Sigma}, C))$. Basically, any state of $\mathcal{A}_R$ makes a guess about the local environment of radius R that it is about to read and then verifies its guess by passing it through a run and checking if other components have made their guess accordingly. A guess is actually an *extended R-sphere* $\sigma = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma, \alpha, \mathit{inst})$ where $\mathit{core}(\sigma) := (V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma) \in \mathcal{S}$ is the pattern that $\mathcal{A}_R$ expects to see, $\alpha \in V$ is the *active vertex*, which corresponds to the vertex that $\mathcal{A}_R$ is about to read, and $\mathit{inst} \in \{1, \ldots, \mathit{const}\}$ is the current *instance* of the pattern where *const* is given by $(2|\Sigma \times C| + 1) \cdot (\max\{|V| \mid (V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma) \in \mathcal{S}\})^2 + 1$. The ACAT $\mathcal{A}_R$, reading some vertex u and entering a state associated with a guess σ, presumes that u is to its local environment as α is to $\mathit{core}(\sigma)$. In other words, $\mathcal{A}_R$ considers u to be the counterpart of α and the environment of u to look like $\mathit{core}(\sigma)$. To establish isomorphism between $\mathit{core}(\sigma)$ and the environment around u, $\mathcal{A}_R$ transfers σ to the immediate successor vertices u' of u except that, in σ, the active vertex α is replaced with some α' such that $\alpha \lhd \alpha'$. This is because u shall correspond to u' only if α corresponds to α'. As a tiling of a graph induces an overlapping of participating spheres, a state of $\mathcal{A}_R$ actually holds a *set* of extended R-spheres, which subsequently have to be forwarded and verified simultaneously. The state q entered when reading u carries exactly one extended R-sphere σ whose sphere center and active vertex coincide with the understanding that the environment of u of radius R is isomorphic to $\mathit{core}(\sigma)$ so that we may set $\eta(q)$ to be $\mathit{core}(\sigma)$. There may even be an overlapping of isomorphic R-spheres so that a state possibly contains several instances of one and the same sphere, which then refer to distinct vertices as corresponding sphere centers. Those instances will be distinguished by means of the natural number *inst*. However, there can be at most *const* such overlappings, an order of magnitude that depends on $(\widetilde{\Sigma}, C)$ and R only.

Table 5.1. Keeping track of spheres by means of an ACAT

Let $(\overline{\mathcal{S}}, b, \mathcal{S}') \in \Delta$ if the following hold:

L $\lambda(\mathcal{S}') = b$.

For any $(a, \ell) \in \Sigma \times C$ with $\overline{\mathcal{S}}[(a, \ell)] \neq -$, any $\sigma \in \overline{\mathcal{S}}[(a, \ell)]$, and any $v \in V$:

W1 If $\sigma[v] \in \mathcal{S}'$, then $\alpha \lhd_\ell v$.
W2 If $(b, \ell) \notin \mathrm{Write}(\alpha)$, then $d(\alpha, \gamma) = R$.
W3 If $(b, \ell) \in \mathrm{Write}(\alpha)$, then $\sigma[(b, \ell)\text{-succ}(\alpha)] \in \mathcal{S}'$.

For any $(a, \ell) \in \Sigma \times C$ and any $\sigma \in \mathcal{S}'$:

R1 If $\overline{\mathcal{S}}[(a, \ell)] \neq -$ and $(a, \ell) \notin \mathrm{Read}(\alpha)$, then $d(\alpha, \gamma) = R$.
R2 If $(a, \ell) \in \mathrm{Read}(\alpha)$, then $\sigma[(a, \ell)\text{-pred}(\alpha)] \in \overline{\mathcal{S}}[(a, \ell)] \neq -$.

Now set $\mathcal{S}^+$ to be the set of extended R-spheres that emerge from $\mathcal{S}$, i.e., $\mathcal{S}^+ := \{(V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma, \alpha, \mathit{inst}) \mid (V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma) \in \mathcal{S},\ \alpha \in V, \text{ and } \mathit{inst} \in \{1, \ldots, \mathit{const}\}\}$. In the scope of extended R-spheres $\sigma, \sigma' \in \mathcal{S}^+$, in the following, we let $V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma, \alpha, \mathit{inst}$ and $V', \{\lhd'_\ell\}_{\ell \in C}, \lambda', \gamma', \alpha', \mathit{inst}'$ refer to the components of σ and σ', respectively. Given $\sigma \in \mathcal{S}^+$ and $v \in V$, $\sigma[v]$ shall denote $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma, v, \mathit{inst}) \in \mathcal{S}^+$, in which the active vertex α of σ is replaced with a new active vertex v.

Let us now turn to the construction of the ACAT $\mathcal{A}_R = (Q, \Delta, T, F)$, which is given as follows: a state from Q is a (nonempty) subset $\mathcal{S}$ of $\mathcal{S}^+$ such that

- there is exactly one extended sphere $\sigma \in \mathcal{S}$ with $\gamma = \alpha$ (we set $\mathit{core}(\mathcal{S})$ to be $\mathit{core}(\sigma)$),
- there is $a \in \Sigma$ such that, for any $\sigma \in \mathcal{S}$, $\lambda(\alpha) = a$ (so that we can assign a well-defined unique label $\lambda(\mathcal{S}) := a$ to $\mathcal{S}$), and
- for any $\sigma, \sigma' \in \mathcal{S}$, if $\mathit{core}(\sigma) = \mathit{core}(\sigma')$ and $\mathit{inst} = \mathit{inst}'$, then $\alpha = \alpha'$.

The definition of Δ is given by Table 5.1. Condition **L** goes without saying. Now assume an extended sphere σ with active vertex α is attached to some u. Assume furthermore that $\sigma[v]$ is attached to some direct successor u' of u. As α and v have to simulate u and u' (and vice versa), they have to be joined by an edge as well. This is what condition **W1** is supposed to guarantee. Suppose now that α lacks a (b, ℓ)-successor, while u does not. That situation is allowed only if the distance from α to γ is R, as then the scope of σ ends anyway so that, beyond u, it is no longer responsible for what will happen (**W2**). Otherwise, if (b, ℓ) is contained in the write domain of α, then σ has to coincide with the input structure further on so that $\sigma[(b, \ell)\text{-succ}(\alpha)]$ is sent to the (b, ℓ)-successor of u (**W3**). The duals of **W2** and **W3**, regarding the read domain of a vertex, are guaranteed by conditions **R1** and **R2**, respectively. Note that condition **W1** lacks its dual case, as this is implicitly present.

Let us now turn to the type function and the set of final states of $\mathcal{A}_R$: for any $a \in \Sigma$ and $\mathcal{S} \in Q$, we set $T(a, \mathcal{S})$ to be $\{(b, \ell) \in (\Sigma, C) \mid$ there is $\sigma \in \mathcal{S}$ such that $(b, \ell) \in \mathrm{Write}(\alpha)\}$, i.e., if the active vertex of some extended sphere from $\mathcal{S}$ has some (b, ℓ)-successor, then so will the vertex to which $\mathcal{S}$ is attached. Finally, we just set F to be $(Q \cup \{\imath\})^{Ag}$. The mapping η as required in the proposition is provided by *core*, i.e., we set $\eta(\mathcal{S}) = \mathit{core}(\mathcal{S})$.

Any Dag Is Accepted

Let $\mathcal{D} = (\widetilde{V}, \{\widetilde{\lhd}_\ell\}_{\ell \in C}, \widetilde{\lambda}) \in \mathbb{DAG}(\widetilde{\Sigma}, C)$. We show that there is an accepting run ρ of $\mathcal{A}_R$ on $\mathcal{D}$ such that, for any $u \in \widetilde{V}$, $\mathit{core}(\rho(u)) = R\text{-Sph}(\mathcal{D}, u)$. In the following, let $\widehat{\rho}$ stand for the mapping $\widetilde{V} \to \mathcal{S}$ that maps a vertex $u \in \widetilde{V}$ onto the R-sphere of $\mathcal{D}$ around u.

First of all, we will distribute instance numbers to each of the involved spheres. There is a mapping $\chi_{\mathcal{D}} : \widetilde{V} \to \{1, \ldots, \mathit{const}\}$ such that, for any $u, u', v, v' \in \widetilde{V}$ with $\widehat{\rho}(u) = \widehat{\rho}(u')$, $u \neq u'$, $d(v, u) \leq R$, and $d(v', u') \leq R$, if $v \mathrel{\widetilde{\lhd}} v'$ or $v' \mathrel{\widetilde{\lhd}} v$ or $v = v'$, then $\chi_{\mathcal{D}}(u) \neq \chi_{\mathcal{D}}(u')$. We can reduce the existence of $\chi_{\mathcal{D}}$ to the existence of a graph coloring: let $\mathcal{G}$ be the graph $(\widetilde{V}, \mathit{Arcs})$ (without labeling function) where, for any $u, u' \in \widetilde{V}$, $(u, u') \in \mathit{Arcs}$ iff $u \neq u'$, $\widehat{\rho}(u) = \widehat{\rho}(u')$, and there is $v, v' \in \widetilde{V}$ with $d(v, u) \leq R$, $d(v', u') \leq R$, and ($v \mathrel{\widetilde{\lhd}} v'$ or $v' \mathrel{\widetilde{\lhd}} v$ or $v = v'$). As $\mathcal{G}$ cannot be of degree greater than $\mathit{const} - 1$ (for each $u \in \widetilde{V}$, there are at most $2|\Sigma \times C| + 1$ distinct vertices $u' \in \widetilde{V}$ such that $u \mathrel{\widetilde{\lhd}} u'$, $u' \mathrel{\widetilde{\lhd}} u$, or $u = u'$), it can be *const*-colored by some mapping $\chi : \widetilde{V} \to \{1, \ldots, \mathit{const}\}$, i.e., $(u, v) \in \mathit{Arcs}$ implies $\chi(u) \neq \chi(v)$ for any $u, v \in \widetilde{V}$. Now just set $\chi_{\mathcal{D}}$ to be χ.

In the scope of extended R-spheres σ and σ' from $\mathcal{S}^+$, we continue to let $V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma, \alpha, \mathit{inst}$ and $V', \{\lhd'_\ell\}_{\ell \in C}, \lambda', \gamma', \alpha', \mathit{inst}'$ refer to the components of σ and σ', respectively. The accepting run $\rho : \widetilde{V} \to Q$ of $\mathcal{A}_R$ on $\mathcal{D}$ is defined as follows: for $u \in \widetilde{V}$, we set $\rho(u) = \{\sigma \in \mathcal{S}^+ \mid$ there is $u' \in \widetilde{V}$ such that $d(u', u) \leq R$, $(\mathit{core}(\sigma), \alpha) = (\widehat{\rho}(u'), u)$, and $\mathit{inst} = \chi_{\mathcal{D}}(u')\}$.

For $u \in \widetilde{V}$, we first verify that, in fact, $\rho(u)$ is a valid state of $\mathcal{A}_R$. So suppose there are extended R-spheres $\sigma, \sigma' \in \rho(u)$. Of course, $\lambda(\alpha) = \lambda'(\alpha')$. Assume now that both $\gamma = \alpha$ and $\gamma' = \alpha'$. But then the requirements $(\mathit{core}(\sigma), \gamma) = (\widehat{\rho}(u), u)$ and $(\mathit{core}(\sigma'), \gamma') = (\widehat{\rho}(u), u)$ imply $(\mathit{core}(\sigma), \gamma) = (\mathit{core}(\sigma'), \gamma')$. In particular, $\mathit{core}(\rho(u)) = \mathit{core}(\sigma) = \widehat{\rho}(u)$. Furthermore, $\mathit{inst} = \mathit{inst}' = \chi_{\mathcal{D}}(u)$. Now assume $\mathit{core}(\sigma) = \mathit{core}(\sigma')$ and $\mathit{inst} = \mathit{inst}'$. There are vertices $u_1, u_2 \in \widetilde{V}$ such that $d(u_1, u) \leq R$, $d(u_2, u) \leq R$, $(\mathit{core}(\sigma), \alpha) = (\widehat{\rho}(u_1), u)$, $(\mathit{core}(\sigma), \alpha') = (\widehat{\rho}(u_2), u)$, and $\mathit{inst} = \chi_{\mathcal{D}}(u_1) = \chi_{\mathcal{D}}(u_2)$. Clearly, we have $\widehat{\rho}(u_1) = \widehat{\rho}(u_2)$. Furthermore, $u_1 = u_2$ and, consequently, $\alpha = \alpha'$. This is because $u_1 \neq u_2$ implies $\chi_{\mathcal{D}}(u_1) \neq \chi_{\mathcal{D}}(u_2)$, which contradicts the premise.

In the following, we verify that ρ is a run of $\mathcal{A}_R$ on $\mathcal{D}$. For any $u \in \widetilde{V}$, we check that $\mathit{trans}_{(\mathcal{D}, \rho)}(u) \in \Delta$. So suppose $\mathit{trans}_{(\mathcal{D}, \rho)}(u) = (\overline{\mathcal{S}}, \widetilde{\lambda}(u), \mathcal{S}')$ for some $\overline{\mathcal{S}}$ and $\mathcal{S}'$.

L Of course, $\lambda(\mathcal{S}') = \widetilde{\lambda}(u)$.

W1 Let $(a,\ell) \in \Sigma \times C$ and $\sigma \in \overline{\mathcal{S}}[(a,\ell)] \neq -$ and suppose there is $v \in V$ such that $\sigma[v] \in \mathcal{S}'$. We set $u^- \in \widetilde{V}$ to be (a,ℓ)-pred(u). There is $u^{-\prime}, u' \in \widetilde{V}$ such that $d(u^{-\prime}, u^-) \leq R$, $d(u', u) \leq R$, $\sigma = (\widehat{\rho}(u^{-\prime}), u^-, \mathit{inst})$, and $\sigma[v] = (\widehat{\rho}(u'), u, \mathit{inst})$ with $\mathit{inst} = \chi_{\mathcal{D}}(u^{-\prime}) = \chi_{\mathcal{D}}(u')$. We show $u^{-\prime} = u'$, as then $\sigma = (\widehat{\rho}(u'), u^-, \mathit{inst})$, $\sigma[v] = (\widehat{\rho}(u'), u, \mathit{inst})$, and $u^- \mathrel{\widetilde{\lhd}} u$ implies $\alpha \lhd v$. But $u^{-\prime} \neq u'$ implies $\mathit{inst} = \chi_{\mathcal{D}}(u^{-\prime}) \neq \chi_{\mathcal{D}}(u')$, which leads to a contradiction to the above assumption.

W2 Let $(a,\ell) \in \Sigma \times C$ and $\sigma \in \overline{\mathcal{S}}[(a,\ell)] \neq -$ with $(\widetilde{\lambda}(u), \ell) \notin \mathrm{Write}(\alpha)$ and set $u^- \in \widetilde{V}$ to be (a,ℓ)-pred(u). As $\overline{\mathcal{S}}[(a,\ell)] = \rho(u^-)$, there is $u^{-\prime} \in \widetilde{V}$ such that both $d(u^{-\prime}, u^-) \leq R$ and $(\mathit{core}(\sigma), \alpha) = (\widehat{\rho}(u^{-\prime}), u^-)$. As $(\widetilde{\lambda}(u), \ell) \in \mathrm{Write}(u^-)$, we have $d(\gamma, \alpha) = d(u^{-\prime}, u^-) = R$.

W3 Let $(a,\ell) \in \Sigma \times C$ and suppose there is $\sigma \in \overline{\mathcal{S}}[(a,\ell)] \neq -$ with $(\widetilde{\lambda}(u), \ell) \in \mathrm{Write}(\alpha)$. Set u^- to be (a,ℓ)-pred(u) and α^+ to be $(\widetilde{\lambda}(u), \ell)$-succ$(\alpha)$. As we have $\overline{\mathcal{S}}[(a,\ell)] = \rho(u^-)$, there exists $u^{-\prime} \in \widetilde{V}$ with $d(u^{-\prime}, u^-) \leq R$, $(\mathit{core}(\sigma), \alpha) = (\widehat{\rho}(u^{-\prime}), u^-)$, and $\mathit{inst} = \chi_{\mathcal{D}}(u^{-\prime})$. Since, then, $d(u^{-\prime}, u) = d(\gamma, \alpha^+) \leq R$ and also $(\mathit{core}(\sigma), \alpha^+) = (\widehat{\rho}(u^{-\prime}), u)$, we have $\sigma[\alpha^+] \in \mathcal{S}'$.

R1 Let $(a,\ell) \in \Sigma \times C$. Suppose $\mathcal{S}[(a,\ell)] \neq -$ and suppose there is $\sigma \in \mathcal{S}'$ with $(a,\ell) \notin \mathrm{Read}(\alpha)$. There is $u' \in \widetilde{V}$ such that $d(u', u) \leq R$ and $(\mathit{core}(\sigma), \alpha) = (\widehat{\rho}(u'), u)$. As $\mathcal{S}[(a,\ell)] \neq -$, we have $(a,\ell) \in \mathrm{Read}(u)$ and, consequently, $d(\gamma, \alpha) = d(u', u) = R$.

R2 Let $(a,\ell) \in \Sigma \times C$ and suppose there is $\sigma \in \mathcal{S}'$ with $(a,\ell) \in \mathrm{Read}(\alpha)$. Let α^- denote (a,ℓ)-pred(α). As $\rho(u) = \mathcal{S}'$, there is $u' \in \widetilde{V}$ with $d(u', u) \leq R$ such that $(\mathit{core}(\sigma), \alpha) = (\widehat{\rho}(u'), u)$ and $\mathit{inst} = \chi_{\mathcal{D}}(u')$. As a consequence, $(a,\ell) \in \mathrm{Read}(u)$ so that there is $u^- \in \widetilde{V}$ with $u^- \mathrel{\widetilde{\lhd}_\ell} u$ and $\widetilde{\lambda}(u^-) = a$. As, furthermore, $d(u', u^-) = d(\gamma, \alpha^-) \leq R$ and $(\mathit{core}(\sigma), \alpha^-) = (\widehat{\rho}(u'), u^-)$, we have $\sigma[\alpha^-] \in \overline{\mathcal{S}}[(a,\ell)] \neq -$.

Moreover, we easily see that ρ is accepting.

Any Run Keeps Track of Spheres

In what follows, we show that, indeed, any accepting run of $\mathcal{A}_R$ tells us about the environment of any node. More precisely, in an accepting run, a state q can be assigned to a node u only if the environment of u looks like $\eta(q)$ (recall that η was determined to be *core*).

Let ρ be an accepting run of $\mathcal{A}_R$ on $\mathcal{D} = (\widetilde{V}, \{\widetilde{\lhd}_\ell\}_{\ell \in C}, \widetilde{\lambda}) \in \mathbb{DAG}(\widetilde{\Sigma}, C)$. We show that, for any $u \in \widetilde{V}$, $\mathit{core}(\rho(u)) = R\text{-Sph}(\mathcal{D}, u)$. In the following, let $\widehat{\rho}$ be the mapping $\widetilde{V} \to \mathcal{S}$ that maps any $u \in \widetilde{V}$ to $\mathit{core}(\rho(u))$, i.e., $\widehat{\rho}$ is given by $\mathit{core} \circ \rho$.

Claim 5.40. For each $u \in \widetilde{V}$, $\sigma \in \rho(u)$, and $d \in \mathbb{N}$, if there is a sequence of vertices $v_0, \ldots, v_d \in V$ such that $v_0 = \alpha$ and, for each $k \in \{0, \ldots, d-1\}$, $v_k \lhd v_{k+1}$ or $v_{k+1} \lhd v_k$, then there is a unique sequence of vertices $u_0, \ldots, u_d \in \widetilde{V}$ with

- $u_0 = u$,
- for each $k \in \{0, \ldots, d\}$, $\sigma[v_k] \in \rho(u_k)$ (and therefore $\lambda(v_k) = \widetilde{\lambda}(u_k)$), and
- for each $k \in \{0, \ldots, d-1\}$ and $\ell \in C$, $u_k \mathrel{\widetilde{\lhd}_\ell} u_{k+1}$ iff $v_k \lhd_\ell v_{k+1}$ and $u_{k+1} \mathrel{\widetilde{\lhd}_\ell} u_k$ iff $v_{k+1} \lhd_\ell v_k$.

Proof of Claim 5.40. We proceed by induction. Obviously, the statement holds for $d = 0$. Now assume there is a sequence of vertices $v_0, \ldots, v_d, v_{d+1} \in V$ such that $v_0 = \alpha$ and, for each $k \in \{0, \ldots, d\}$, $v_k \lhd v_{k+1}$ or $v_{k+1} \lhd v_k$. By the induction hypothesis, there is a unique sequence of vertices $u_0, \ldots, u_d \in \widetilde{V}$ satisfying the above conditions. Let $\ell \in C$ and suppose that

- $v_d \lhd_\ell v_{d+1}$. As then $(\lambda(v_{d+1}), \ell) \in \mathrm{Write}(v_d)$ and due to the definition of the type function of $\mathcal{A}_R$, there must be a unique vertex $u_{d+1} \in \widetilde{V}$ with $u_d \mathrel{\widetilde{\lhd}_\ell} u_{d+1}$ and $\widetilde{\lambda}(u_{d+1}) = \lambda(v_{d+1})$. Furthermore, due to condition **W3**, we have $\sigma[v_{d+1}] \in \rho(u_{d+1})$.
- $v_{d+1} \lhd_\ell v_d$. Then, there is, according to condition **R2**, a unique $u_{d+1} \in \widetilde{V}$ with $u_{d+1} \mathrel{\widetilde{\lhd}_\ell} u_d$ and $\sigma[v_{d+1}] \in \rho(u_{d+1})$. This shows Claim 5.40. □

We have to prove that, for each $u \in \widetilde{V}$, the R-sphere of $(\widetilde{V}, \{\widetilde{\lhd}_\ell\}_{\ell \in C}, \widetilde{\lambda})$ around u is isomorphic to $\widehat{\rho}(u)$. So let $u \in \widetilde{V}$, set $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma)$ to be $\widehat{\rho}(u)$, and let $inst \in \{1, \ldots, const\}$ be the unique element with $\sigma := (V, \{\lhd_\ell\}_{\ell \in C}, \lambda, \gamma, \gamma, inst) \in \rho(u)$.

Claim 5.41. For each $d \in \{0, \ldots, R\}$, there is an isomorphism

$$h : d\text{-Sph}((\widetilde{V}, \{\widetilde{\lhd}_\ell\}_{\ell \in C}, \widetilde{\lambda}), u) \to d\text{-Sph}((V, \{\lhd_\ell\}_{\ell \in C}, \lambda), \gamma)$$

such that, for any $u' \in \widetilde{V}$ with $d(u', u) \leq d$, $\sigma[h(u')] \in \rho(u')$.

Proof of Claim 5.41. We proceed by induction. Of course, the statement holds for $d = 0$. Now assume that $d < R$ and that there is an isomorphism $h : d\text{-Sph}((\widetilde{V}, \{\widetilde{\lhd}_\ell\}_{\ell \in C}, \widetilde{\lambda}), u) \to d\text{-Sph}((V, \{\lhd_\ell\}_{\ell \in C}, \lambda), \gamma)$ such that, for each $u' \in \widetilde{V}$ with $d(u', u) \leq d$, $\sigma[h(u')] \in \rho(u')$.

Extended Sphere Simulates Dag

Suppose there is $u_1, u_1', u_2, u_2' \in \widetilde{V}$ such that $d(u, u_1) = d(u, u_2) = d$, $d(u, u_1') = d(u, u_2') = d + 1$, ($u_1 \mathrel{\widetilde{\lhd}} u_1'$ or $u_1' \mathrel{\widetilde{\lhd}} u_1$), and ($u_2 \mathrel{\widetilde{\lhd}} u_2'$ or $u_2' \mathrel{\widetilde{\lhd}} u_2$). Given $\ell \in C$, suppose (let v_1 and v_2 denote $h(u_1)$ and $h(u_2)$, respectively)

- $u_1 \mathrel{\widetilde{\lhd}_\ell} u_1'$. As $d(u, u_1) < R$, we have $d(\gamma, v_1) < R$. Due to **W2**, there is $v_1' \in V$ with $v_1 \lhd_\ell v_1'$ and $\lambda(v_1') = \widetilde{\lambda}(u_1')$ and, due to **W3** and $\sigma[v_1] \in \rho(u_1)$, $\sigma[v_1'] \in \rho(u_1')$.
- $u_1' \mathrel{\widetilde{\lhd}_\ell} u_1$. As $d(u, u_1)$ is less than R, so is $d(\gamma, v_1)$. Due to **R1**, there is $v_1' \in V$ with $v_1' \lhd_\ell v_1$ and $\lambda(v_1') = \widetilde{\lambda}(u_1')$ and, due to **R2** and $\sigma[v_1] \in \rho(u_1)$, $\sigma[v_1'] \in \rho(u_1')$.

Thus, depending on u_1', we obtain from v_1 a unique vertex $v_1' \in V$, which we denote by $h'(u_1')$. According to the above scheme, we obtain from v_2 a unique vertex $v_2' \in V$, denoted by $h'(u_2')$. Then $d(v_1', \gamma) = d(v_2', \gamma) = d + 1$. Now suppose, given $\ell \in C$,

- $u_1' \mathrel{\widetilde{\lhd}_\ell} u_2'$. As $\sigma[v_1'] \in \rho(u_1')$ and $\sigma[v_2'] \in \rho(u_2')$, it follows from **W1** that $v_1' \lhd_\ell v_2'$.
- $u_1' = u_2'$. Then, $\sigma[v_1'] \in \rho(u_1')$ and $\sigma[v_2'] \in \rho(u_1')$ implies $v_1' = v_2'$.

The case $u_2' \mathrel{\widetilde{\lhd}_\ell} u_1'$ can be handled analogously and the cases $u_1' \mathrel{\widetilde{\not\lhd}_\ell} u_2'$ and $u_1' \neq u_2'$ as below.

Dag Simulates Extended Sphere

Suppose there is $v_1, v_1', v_2, v_2' \in V$ such that $d(v_1, \gamma) = d(v_2, \gamma) = d$, $d(v_1', \gamma) = d(v_2', \gamma) = d + 1$, ($v_1 \lhd v_1'$ or $v_1' \lhd v_1$), and ($v_2 \lhd v_2'$ or $v_2' \lhd v_2$). We now proceed as in the proof of Claim 5.40. So let $\ell \in C$ and suppose (let u_1 and u_2 denote $h^{-1}(v_1)$ and $h^{-1}(v_2)$, respectively)

- $v_1 \lhd_\ell v_1'$. Then, there is $u_1' \in \widetilde{V}$ with $u_1 \mathrel{\widetilde{\lhd}_\ell} u_1'$ and, due to **W3**, $\sigma[v_1'] \in \rho(u_1')$.
- $v_1' \lhd_\ell v_1$. According to **R2**, there is $u_1' \in \widetilde{V}$ with $u_1' \mathrel{\widetilde{\lhd}_\ell} u_1$ and $\sigma[v_1'] \in \rho(u_1')$.

According to the above scheme, we obtain from u_2 a unique vertex u_2'. Now let $\ell \in C$ and suppose

- $v_1' \lhd_\ell v_2'$. Assume $u_1' \mathrel{\widetilde{\not\lhd}_\ell} u_2'$. According to the definition of the set of states of $\mathcal{A}_R$, $v_1' \neq v_2'$, $\sigma[v_1'] \in \rho(u_1')$, and $\sigma[v_2'] \in \rho(u_2')$ imply $u_1' \neq u_2'$. But then, following the scheme depicted in Fig. 5.11, we can construct an infinite sequence $\widehat{u}_1, \widehat{u}_2, \ldots \in \widetilde{V}$ inducing an infinite set of (pairwise distinct) vertices: Set $\widehat{u}_1 \in \widetilde{V}$ to be the unique $\lambda(v_2')$-labeled vertex satisfying $u_1' \mathrel{\widetilde{\lhd}_\ell} \widehat{u}_1$. We have $\sigma[v_2'] \in \rho(\widehat{u}_1)$. Suppose $\widehat{u}_1 \mathrel{\widetilde{<}} u_2'$ (the other case is handled analogously). According to Claim 5.40, there is $\widehat{u}_2 \in \widetilde{V}$ such that $\sigma \in \rho(\widehat{u}_2)$ and $\widehat{u}_2 \mathrel{\widetilde{<}} u$. (There is a path in $(V, \{\lhd_\ell\}_{\ell \in C}, \lambda)$ from v_2' to γ that, according to Claim 5.40, takes $\mathcal{D}$ from u_2' to u. Apply this path to $\widehat{u}_1$ yielding a path to a unique vertex $\widehat{u}_2 \in \widetilde{V}$ with $\sigma \in \rho(\widehat{u}_2)$. From $\widehat{u}_1 \mathrel{\widetilde{<}} u_2'$ and the definition of a $(\widetilde{\Sigma}, C)$-dag, it easily follows that $\widehat{u}_2 \mathrel{\widetilde{<}} u$.) Similarly, there is $\widehat{u}_3 \in \widetilde{V}$ with $\sigma[v_1'] \in \rho(\widehat{u}_3)$ and $\widehat{u}_3 \mathrel{\widetilde{<}} u_1'$. Now let $\widehat{u}_4 \in \widetilde{V}$ be the unique vertex such that $\widehat{u}_3 \mathrel{\widetilde{\lhd}_\ell} \widehat{u}_4$ and $\sigma[v_2'] \in \rho(\widehat{u}_4)$ and let, again following Claim 5.40, $\widehat{u}_5 \in \widetilde{V}$ be a vertex with $\sigma \in \rho(\widehat{u}_5)$ and $\widehat{u}_5 \mathrel{\widetilde{<}} \widehat{u}_2$, and $\widehat{u}_6 \in \widetilde{V}$ be a vertex with $\sigma[v_1'] \in \rho(\widehat{u}_6)$ and $\widehat{u}_6 \mathrel{\widetilde{<}} \widehat{u}_3$. Continuing this scheme yields an infinite set of vertices, contradicting the premise that we deal with finite structures.
- $v_1' = v_2'$. Again, assuming $u_1' \neq u_2'$, we generate a sequence $\widehat{u}_1, \widehat{u}_2, \ldots \in \widetilde{V}$ inducing an infinite set of vertices as follows: suppose $u_1' \mathrel{\widetilde{<}} u_2'$. According to Claim 5.40, we can find $\widehat{u}_1 \in \widetilde{V}$ such that $\sigma \in \rho(\widehat{u}_1)$ and $\widehat{u}_1 \mathrel{\widetilde{<}} u$. Furthermore, there is $\widehat{u}_2 \in \widetilde{V}$ satisfying $\sigma[v_1'] \in \rho(\widehat{u}_2)$ and $\widehat{u}_2 \mathrel{\widetilde{<}} u_1'$ and so on.

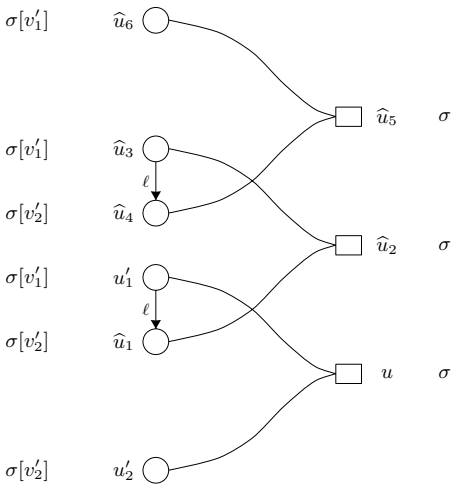


Fig. 5.11. An infinite sequence of vertices

The other cases are handled analogously. From the above results, we conclude that the mapping

$$\widehat{h} : (d+1)\text{-Sph}((\widetilde{V}, \{\widetilde{\lhd}_\ell\}_{\ell\in C}, \widetilde{\lambda}), u) \to (d+1)\text{-Sph}((V, \{\lhd_\ell\}_{\ell\in C}, \lambda), \gamma)$$

with $\widehat{h}(w) = h(w)$ if $d(w,u) \leq d$ and $\widehat{h}(w) = h'(w)$ if $d(w,u) = d+1$ (for $w \in \widetilde{V}$ with $d(w,u) \leq d+1$) is an isomorphism satisfying, for any $w \in \widetilde{V}$ with $d(w,u) \leq d+1$, $\sigma[\widehat{h}(w)] \in \rho(w)$. This concludes the proof of Claim 5.41. □
Altogether, we have shown Proposition 5.39. □

Example 5.42. In the following, let $\mathcal{H}$ denote the 2-sphere in part (a) from Fig. 3.1 on page 23 (neglecting the edge labelings). Figure 5.12, showing some $((\{a\},\{b\},\{c\},\{d\}),-)$-dag $\mathcal{D}$, illustrates the transition behavior of the ACAT $\mathcal{A}_R$ from the proof of Proposition 5.39. It demonstrates how a run of $\mathcal{A}_R$ on $\mathcal{D}$ transfers extensions of $\mathcal{H}$ from one vertex of $\mathcal{D}$ to a neighboring one to make sure that the 2-sphere around u_c (which is indicated by solid edges) is isomorphic to $\mathcal{H}$. For example, the state that is taken on event u_a may contain the extended sphere $(\mathcal{H}, \mathrm{a})$. (For clarity, control states and the natural number *inst* to distinguish different instances of spheres are omitted.) As $\mathrm{a} \lhd \mathrm{b}$ (with respect to the edge relation of $\mathcal{H}$), the type function of $\mathcal{A}_R$ makes sure that u_a has a b-labeled successor vertex u_b, which, by **W3**, must carry $(\mathcal{H}, \mathrm{b})$. Again, u_b cannot be part of a global final state and is followed by some u_c, which has to be associated with a state containing $(\mathcal{H}, \mathrm{c})$ (**W3**). In contrast, u_h is not allowed to carry $(\mathcal{H}, \mathrm{e})$, unless it belongs to a different instance of $\mathcal{H}$ (**W1**). Now consider u_d, which holds the extended sphere $(\mathcal{H}, \mathrm{d})$. Due to **R2**, the preceding state that is associated to u_c must contain $(\mathcal{H}, \mathrm{c})$, which means

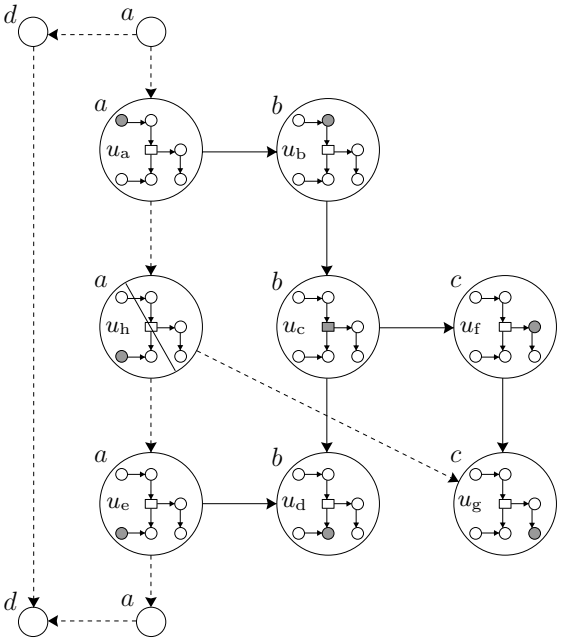


Fig. 5.12. The transitions of $\mathcal{A}_R$ to simulate a graph acceptor

that a run cannot simply enter $\mathcal{H}$ beginning with d. The same applies to u_d, which must contain $(\mathcal{H}, \mathrm{d})$. Note that, as $d(\mathrm{a}, \mathrm{c}) = d(\mathrm{e}, \mathrm{c}) = 2$, the (illustrated parts of the) states assigned to u_a and u_e satisfy **W2** and **R1**.

Lemma 5.43. *For any $t \in \mathbb{N}$ and any $\alpha \in \mathit{Cond}(\Sigma, t)$, there is an ACA $\mathcal{A}_\alpha$ over $(\widetilde{\Sigma}, C)$ such that, for any $\mathcal{D} \in \mathbb{DAG}(\widetilde{\Sigma}, C)$, we have $\mathcal{D} \in L(\mathcal{A}_\alpha)$ iff $\mathcal{D} \models \alpha$.*

Proof. We specify $\mathcal{A}_\alpha = (Q, \Delta, T, F)$ as follows. Let $Q = \{q \mid q : \Sigma \to \{0, \ldots, t\}\}$. Of course, $T(a, q) = \emptyset$ for any $(a, q) \in \Sigma \times Q$. Moreover,

$$\{((a_1, q_1), \ell_1), \ldots, ((a_n, q_n), \ell_n)\} \longrightarrow (b, q) \in \mathit{Trans}_{(\widetilde{\Sigma}, C)}(Q)$$

is contained in Δ if, for any $a \in \Sigma$, $q(a)$ is the minimum of t and

$$\begin{cases} \max(\{0\} \cup \{q_k(a) \mid k \in \{1, \ldots, n\}\}) & \text{if } a \neq b \\ \max(\{1\} \cup \{q_k(a) + 1 \mid k \in \{1, \ldots, n\}\}) & \text{if } a = b \end{cases}$$

Given $\overline{q} \in (Q \cup \{\imath\})^{Ag}$, we define the mapping $f_{\overline{q}} : \Sigma \to \{0, \ldots, t\}$ and, for $a \in \Sigma$, set $f_{\overline{q}}(a)$ to be $\max(\{0\} \cup \{\overline{q}[i](a) \mid i \in Ag, \overline{q}[i] \neq \imath\})$. Finally, we may set $F = \{\overline{q} \in (Q \cup \{\imath\})^{Ag} \mid f_{\overline{q}} \models \alpha\}$. □

Lemma 5.44.

$$\mathcal{EMSO}(\Sigma, C)_{\mathbb{DAG}(\widetilde{\Sigma}, C)} \subseteq \mathcal{ACAT}(\widetilde{\Sigma}, C)$$

Proof. Suppose φ to be an EMSO(Σ, C)-sentence. According to Theorem 3.24, there is a graph acceptor $\mathcal{B} = (Q, R, \mathcal{S}, Occ)$ over (Σ, C) (with $Occ \in Cond(\mathcal{S}, t)$ for some $t \in \mathbb{N}$) such that $L_{\mathbb{DAG}(\widetilde{\Sigma}, C)}(\mathcal{B}) = L_{\mathbb{DAG}(\widetilde{\Sigma}, C)}(\varphi)$. Our aim is to transform $\mathcal{B}$ into an ACAT $\mathcal{A}$ over $(\widetilde{\Sigma}, C)$ such that $L(\mathcal{A}) = L_{\mathbb{DAG}(\widetilde{\Sigma}, C)}(\mathcal{B})$.

We define a distributed alphabet $\langle \widetilde{\Sigma}, Q\rangle := (\Sigma_i \times Q)_{i \in Ag}$. Note that $\langle \mathbb{DAG}(\widetilde{\Sigma}, C), Q\rangle \subseteq \mathbb{DAG}(\langle \widetilde{\Sigma}, Q\rangle, C)$. From Proposition 5.39, we know that we can assume the existence of an ACAT $\mathcal{A}_R = (Q_R, \Delta_R, T_R, F_R)$ over $(\langle \widetilde{\Sigma}, Q\rangle, C)$ and a mapping $\eta : Q_R \to R\text{-Sph}(\mathbb{DAG}(\langle \widetilde{\Sigma}, Q\rangle, C))$ such that $L(\mathcal{A}_R) = \mathbb{DAG}(\langle \widetilde{\Sigma}, Q\rangle, C)$ and, for any $(\langle \widetilde{\Sigma}, Q\rangle, C)$-dag $\mathcal{D} = (V, \{\lhd_\ell\}_{\ell \in C}, \lambda)$, any accepting run ρ of $\mathcal{A}_R$ on $\mathcal{D}$, and any $u \in V$, we have $\eta(\rho(u)) = R\text{-Sph}(\mathcal{D}, u)$. From $\mathcal{A}_R$, we first construct an ACAT $\mathcal{A}' = (Q', \Delta', T', F')$ over $(\langle \widetilde{\Sigma}, Q\rangle, C)$ such that $L(\mathcal{A}') = \{\mathcal{D} \in \mathbb{DAG}(\langle \widetilde{\Sigma}, Q\rangle, C) \mid \mathcal{D} \models Occ$ and $|\mathcal{D}|_{\mathcal{H}} = 0$ for any $\mathcal{H} \in R\text{-Sph}(\mathbb{DAG}(\langle \widetilde{\Sigma}, Q\rangle, C)) \setminus \mathcal{S}\}$. Here, we proceed similarly to the proof of Lemma 5.43 to implement a threshold counting procedure. So let Q' be the cartesian product of Q_R and $\{q \mid q : \mathcal{S} \to \{0, \ldots, t\}\}$. We set $T'((a,p),(\mathcal{S},q)) = T_R((a,p),\mathcal{S})$ for any $(a,p) \in \Sigma \times Q$ and $(\mathcal{S},q) \in Q'$. Moreover,

$$\{(((a_1,p_1),(\mathcal{S}_1,q_1)),\ell_1),\ldots,(((a_n,p_n),(\mathcal{S}_n,q_n)),\ell_n)\} \longrightarrow ((b,p),(\mathcal{S},q))$$

from $Trans_{(\langle \widetilde{\Sigma}, Q\rangle, C)}(Q')$ is contained in Δ' if

$$\{(((a_1,p_1),\mathcal{S}_1),\ell_1),\ldots,(((a_n,p_n),\mathcal{S}_n),\ell_n)\} \longrightarrow ((b,p),\mathcal{S}) \in \Delta_R,$$

$\eta(\mathcal{S}) \in \mathcal{S}$, and, for any $\mathcal{H} \in \mathcal{S}$, $q(\mathcal{H})$ is the minimum of t and

$$\begin{cases} \max(\{0\} \cup \{q_k(\mathcal{H}) \mid k \in \{1,\ldots,n\}\}) & \text{if } \mathcal{H} \neq \eta(\mathcal{S}) \\ \max(\{1\} \cup \{q_k(\mathcal{H}) + 1 \mid k \in \{1,\ldots,n\}\}) & \text{if } \mathcal{H} = \eta(\mathcal{S}) \end{cases}$$

Given $\overline{q} \in (Q' \cup \{\imath\})^{Ag}$, we define the mapping $f_{\overline{q}} : \mathcal{S} \to \{0,\ldots,t\}$ by setting, for $\mathcal{H} \in \mathcal{S}$, $f_{\overline{q}}(\mathcal{H})$ to be $\max(\{0\} \cup \{q(\mathcal{H}) \mid i \in Ag, \overline{q}[i] = (\mathcal{S},q) \neq \imath\})$. Finally, we may set $F' = \{\overline{q} \in (Q' \cup \{\imath\})^{Ag} \mid f_{\overline{q}} \models Occ\}$.

It is now easy to construct from $\mathcal{A}'$ the above-mentioned ACAT $\mathcal{A}$ over $(\widetilde{\Sigma}, C)$ (with set of states $Q' \times Q$) to recognize $h(L(\mathcal{A}')) \cap \mathbb{DAG}(\widetilde{\Sigma}, C)$, which equals $L_{\mathbb{DAG}(\widetilde{\Sigma}, C)}(\mathcal{B})$. □

The following theorem now follows from Lemmata 5.38 and 5.44.

Theorem 5.45.

$$\mathcal{ACAT}(\widetilde{\Sigma}, C) = \mathcal{EMSO}(\Sigma, C)_{\mathbb{DAG}(\widetilde{\Sigma}, C)}$$

Both conversions, from automata to formulas and vice versa, are effective.

Proof. It remains to argue that the transformation of an EMSO sentence into a graph acceptor is effective. This is indeed the case, because converting φ into a graph acceptor is effective: according to Theorem 3.15, a radius R and a threshold t can be computed so that $L_{\mathbb{DAG}(\widetilde{\Sigma},C)}(\varphi)$ is the finite union of $\leftrightarrows_{R,t}$-equivalence classes, which do not distinguish graphs in which any sphere of radius R appears more than $t-1$ times or equally often. Recall that, in turn, the equivalence classes of $\leftrightarrows_{R,t}$ can be captured by a graph acceptor. □

Remark 5.46. Recall that the ACAT $\mathcal{A}_R$, which we constructed in the proof of Proposition 5.39, relies on instances of spheres. Instances are indispensable even in the context of MSCs. Consider Fig. 5.13, which depicts an MSC inducing two isomorphic spheres, say of type $\mathcal{H}$. Obviously, α' is actually not allowed to carry $\mathcal{H}$ forward. As the example shows, however, both α and α' must be able to carry distinct copies of $\mathcal{H}$ 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 reader may, however, find classes of dags where instances of spheres can be omitted.

Remark 5.47. In contrast to the setting of MSO formulas and finite automata (cf. Chap. 4), the transformation of an EMSO formula into an equivalent ACAT has elementary complexity. Recall that, according to Theorem 3.15, the radius R computed from the length of an EMSO formula $\varphi = \exists X_1 \ldots \exists X_n \varphi'$ with first-order kernel φ' can be assumed to be $3^{|\varphi|}$. Moreover, set Q to be $\{0,1\}^n$. The state space Q_R of the ACAT $\mathcal{A}_R$ over $(\langle\widetilde{\Sigma}, Q\rangle, C)$ is composed of subsets of the set of extended spheres that arise from R-Sph$(\mathbb{DAG}(\langle\widetilde{\Sigma}, Q\rangle, C))$. The construction of an ACAT $\mathcal{A}$ for φ from the proof of Lemma 5.44 then adds a component to Q_R that counts the elements from R-Sph$(\mathbb{DAG}(\langle\widetilde{\Sigma}, Q\rangle, C))$ up to a threshold $|\varphi| \cdot c$ where c is the maximal size of an R-sphere from R-Sph$(\mathbb{DAG}(\langle\widetilde{\Sigma}, Q\rangle, C))$. The reader may verify that the maximal element has no more than $R^{2\cdot|\Sigma\times C|}$ nodes. Clearly, the number of states of $\mathcal{A}$ can be bounded by an elementary function of $|\varphi|$ (and $|\Sigma \times C|$).

Exercise 5.48. Depending on $\widetilde{\Sigma}$, C (Ag, respectively), and R, provide small upper bounds for the size of the following sets:

(a) R-Sph$(\mathbb{DAG}(\widetilde{\Sigma}, C))$,
(b) R-Sph$(\mathbb{MSC}(Ag))$,
(c) R-Sph$(\mathbb{TR}^+(\widetilde{\Sigma}))$,
(d) R-Sph$(\mathbb{TR}^-(\widetilde{\Sigma}))$,
(e) R-Sph$(\mathbb{W}(\Sigma))$.

We can now exhibit many classes of graphs that are relevant for describing the behavior of distributed systems where the use of patterns of radius 1 is sufficient. This important property is shared by the domains of words, traces, trees, and grids [92].

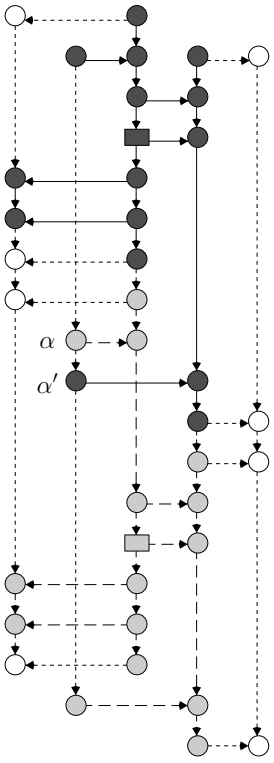


Fig. 5.13. Why we need different instances of extended spheres

Theorem 5.49.

$$\mathcal{GA}(\Sigma, C)_{\mathbb{DAG}(\widetilde{\Sigma},C)} = 1\text{-}\mathcal{GA}(\Sigma, C)_{\mathbb{DAG}(\widetilde{\Sigma},C)}$$

Proof. Let $\mathcal{B}$ be a graph acceptor over (Σ, C). According to Theorems 3.24 and 5.45, there is an ACAT $\mathcal{A} = (Q, \Delta, T, F)$ over $(\widetilde{\Sigma}, C)$ such that $L(\mathcal{A}) = L_{\mathbb{DAG}(\widetilde{\Sigma},C)}(\mathcal{B})$. Without loss of generality, we shall assume that, for any $\overline{q} \in F$ and $i \in Ag$ with $\overline{q}[i] \in Q$, $\overline{q}[i]$ can be assigned at most to the Σ_i-maximal vertex of a $(\widetilde{\Sigma}, C)$-dag (cf. Exercise 5.25). A graph acceptor $\mathcal{B}'$ of radius 1 with $L_{\mathbb{DAG}(\widetilde{\Sigma},C)}(\mathcal{B}') = L(\mathcal{A})$ is given by $(Q, 1, \mathcal{S}, Occ)$ where, for any transition $\{((a_1, p_1), \ell_1), \ldots, ((a_n, p_n), \ell_n)\} \longrightarrow (b, q) \in \Delta$ and any $\{((c_1, q_1), \ell'_1), \ldots, ((c_m, q_m), \ell'_m)\} \subseteq (\Sigma \times Q) \times C$ with $(c_i, \ell'_i) \neq (c_j, \ell'_j)$ $(i \neq j)$ and $T(b, q) \subseteq \{(c_1, \ell'_1), \ldots, (c_m, \ell'_m)\}$, $\mathcal{S}$ contains $\mathcal{H}$ iff removing from $\mathcal{H}$ the edges that do not touch its center yields $\{((a_1, p_1), \ell_1), \ldots, ((a_n, p_n), \ell_n)\} \longrightarrow (b, q) \longrightarrow \{((c_1, q_1), \ell'_1), \ldots, ((c_m, q_m), \ell'_m)\}$ (with the expected meaning).

Now, given $q \in Q$, let $\mathcal{S}_q$ contain those spheres whose sphere center is labeled with (a, q) for some a, and let, accordingly, $\mathcal{S}_a$ with $a \in \Sigma$ contain those spheres whose sphere center is labeled with (a, q) for some q. Then,

$$Occ = \bigvee_{\overline{q}\in F} \Big(\bigwedge_{i\in Ag,\ \overline{q}[i]\in Q} \ \bigvee_{\mathcal{H}\in\mathcal{S}_{\overline{q}[i]}} \mathcal{H} \geq 1 \wedge \bigwedge_{i\in Ag,\ \overline{q}[i]=\iota,\ a\in\Sigma_i,\ \mathcal{H}\in\mathcal{S}_a} \neg(\mathcal{H} \geq 1) \Big)$$

guarantees that a run of $\mathcal{B}$ is accepting only if the corresponding run of $\mathcal{A}$ is accepting. □

Note that, in the context of pictures, a corresponding reduction has been applied to tiling systems [38].

The following theorems will be a consequence of a later result in the framework of MSCs (cf. Theorems 9.10 and 9.12, Lemma 8.26, and Exercises 5.36 and 9.13).

Theorem 5.50. *In general, we have*

$$\mathcal{EMSO}_{\mathbb{DAG}(\widetilde{\Sigma},C)} \subsetneqq \mathcal{MSO}_{\mathbb{DAG}(\widetilde{\Sigma},C)}$$

In particular, there are a distributed alphabet $\widetilde{\Sigma}$ and an alphabet C such that, for some language $L \in \mathcal{ACAT}(\widetilde{\Sigma}, C)$, the complement $\mathbb{DAG}(\widetilde{\Sigma}, C) \setminus L$ of L is however not contained in $\mathcal{ACAT}(\widetilde{\Sigma}, C)$.

Theorem 5.51. *In general, $\mathcal{ACAT}(\widetilde{\Sigma}, C)$ is not closed under complementation.*

Theorem 5.52. *In general, we have*

$$\text{det-}\mathcal{ACAT}(\widetilde{\Sigma}, C) \subsetneqq \mathcal{ACAT}(\widetilde{\Sigma}, C)$$

Checking automata models for emptiness is a crucial verification task. In most cases, however, there is no algorithm that takes an ACAT as the input and decides whether it recognizes the empty language or not. However, we can restrict to certain subclasses of $\mathbb{DAG}(\widetilde{\Sigma}, C)$ or to ACAs to gain decidability of the emptiness problem.

Theorem 5.53.

1. *The following problem is decidable:*

   Input: ACA $\mathcal{A}$ over $\widetilde{\Gamma}(Ag)$.
   Question: Is $L_{\mathbb{LMSC}(Ag)}(\mathcal{A})$ empty?

2. *The following problem is decidable:*

   Input: ACA $\mathcal{A}$ over $(\{a\})_{a\in\Sigma}$.
   Question: Is $L_{\mathbb{DAG}((\{a\})_{a\in\Sigma})}(\mathcal{A})$ empty?

3. *In general, the following problem is undecidable:*

   Input: ACAT $\mathcal{A}$ over $(\widetilde{\Sigma}, C)$.
   Question: Is $L_{\mathbb{DAG}(\widetilde{\Sigma},C)}(\mathcal{A})$ empty?

4. The following problem is undecidable:

> *Input: ACA $\mathcal{A}$ over $\widetilde{\Gamma}(Ag)$.*
> *Question: Is $L_{\mathbb{MSC}(Ag)}(\mathcal{A})$ empty?*

5. The following problem is undecidable:

> *Input: ACAT $\mathcal{A}$ over $\widetilde{\Gamma}(Ag)$.*
> *Question: Is $L_{\mathbb{LMSC}(Ag)}(\mathcal{A})$ empty?*

Proof. Statement 1. follows from [1], 2. has been shown by Kuske in [55]. Result 4. will be shown in Chap. 8, whereas statements 3. and 5. are reductions from 4., which may be verified by the reader. □

5.6 Summary

Some closure and expressiveness properties of ACATs are summarized by Table 5.2 relative to several classes of $(\widetilde{\Sigma}, C)$-dags. In particular, ACATs running over the classes of traces and words adopt the nice properties of finite automata. Things become more complicated when considering classes such as LMSCs or MSCs. Here, EMSO no longer captures the full expressive power of MSO logic so that we lose closure under complementation and have to make do with a weaker deterministic automata model. However, recall that emptiness is still decidable concerning ACAs running over LMSCs. Note that some of the results exhibited in Table 5.2 will be shown in the course of this book.

Table 5.2. Closure and expressiveness properties of asynchronous cellular automata with types

	$\cup$	$\cap$	$\overline{\cdot}$	ACA	det	EMSO	MSO	Empt.
$\mathcal{ACAT}_{\mathbb{DAG}(\widetilde{\Sigma},C)}$	✓	✓	✗	$\supsetneq$	$\supsetneq$	$=$	$\subsetneq$	✗
$\mathcal{ACAT}_{\mathbb{DAG}_{\rightrightarrows}(\widetilde{\Sigma},C)}$	✓	✓	✗	$\supsetneq$	$\supsetneq$	$=$	$\subsetneq$	✗
$\mathcal{ACAT}_{\mathbb{MSC}(Ag)}$	✓	✓	✗	$=$	$\supsetneq$	$=$	$\subsetneq$	✗
$\mathcal{ACAT}_{\mathbb{LMSC}(Ag)}$	✓	✓	✗	$\supsetneq$	$\supsetneq$	$=$	$\subsetneq$	✗
$\mathcal{ACAT}_{\mathbb{TR}^+(\widetilde{\Sigma})}$	✓	✓	✓	$=$	$=$	$=$	$=$	✓
$\mathcal{ACAT}_{\mathbb{TR}^-(\widetilde{\Sigma})}$	✓	✓	✓	$=$	$=$	$=$	$=$	✓
$\mathcal{ACAT}_{\mathbb{W}(\Sigma)}$	✓	✓	✓	$=$	$=$	$=$	$=$	✓

5.7 Bibliographic Notes

ACAs (without types) were introduced originally by Zielonka in the framework of Mazurkiewicz traces [97]. They were generalized by Droste, Gastin, and Kuske to run on posets without autoconcurrency [29]; they also showed that ACAs are expressively equivalent to EMSO[$\leq$]-logic relative to *CROW-posets*, which are subject to an axiom that considers concurrent read and exclusive owner write, and relative to certain *k-posets*, which are more general than CROW-posets but involve a weaker deterministic automata model. Rather than graphs, Droste et al. consider posets and distinguish two reading modes of ACAs, R^+ and R^-, which are also covered in our setting and precisely correspond to the modes M^+ and M^- when considering Mazurkiewicz traces. ACAs running on graphs were studied first in [55]. A way to cope with posets with autoconcurrency in the realm of regular and recognizable languages is presented in [33].

Similarly to ACAs, a *vertex-marking graph automaton*, as introduced in [84], collects states it has already read and thereupon assigns a new state to a common successor vertex. Following the idea of communication requests, an acceptance condition in terms of final states is implicitly given by the requirement that any event has to conform to the type of its state. Vertex-marking graph automata also appear as a special case of Thomas' graph acceptors with radius 1. The reduction of the radius of graph acceptors for the domains of words, traces, trees, and grids is addressed in [92].

ACATs were introduced in [14][3], which also provides the technique we apply for proving that ACATs correspond to EMSO logic relative to any class of $(\widetilde{\Sigma}, C)$-dags. That technique generalized the one from [17] to characterize the class of communicating finite-state machines in terms of EMSO logic.

[3] There, it was claimed that ACATs have already the full expressiveness if a set $T(a, q)$ may only contain pairs (b, ℓ) with b independent of a. While this applies to "reasonable" classes such as M^+-traces, M^--traces, and message sequence charts (in particular, to locally covering classes), the model is actually not sufficient for arbitrary classes of $(\widetilde{\Sigma}, C)$-dags. In fact, to achieve the full expressiveness of EMSO logic, $T(a, q)$ must be able to range over $2^{\Sigma \times C}$.

6

Mazurkiewicz Traces and Asynchronous Automata

We will now study Mazurkiewicz traces, which have been touched on in the previous chapter, in more detail. Mazurkiewicz traces are suitable to model communication where components synchronize by executing certain actions simultaneously, whereas others are taken autonomously.

6.1 Mazurkiewicz Traces

Recall that Mazurkiewicz traces are classified as M^+-traces or M^--traces:

Definition 6.1 (M^+-Trace). *An* M^+-trace *over* $\widetilde{\Sigma}$ *is a dag* $(V, \{\lhd_\ell\}_{\ell \in 2^{Ag}}, \lambda)$ *from* $\mathbb{DAG}(\widetilde{\Sigma}, 2^{Ag})$ *such that*

- $\lhd = \bigcup_{i \in Ag} \lessdot_i$, *and*
- *for any* $(u, v) \in \lhd$ *and* $\ell \in 2^{Ag}$, $u \lhd_\ell v$ *iff* $\ell = \{i \in Ag \mid u \lessdot_i v\}$.

Definition 6.2 (M^--Trace). *An* M^--trace *over* $\widetilde{\Sigma}$ *is a* $\widetilde{\Sigma}$*-dag* $(V, \lhd, \lambda) \in \mathbb{DAG}_H(\widetilde{\Sigma})$ *such that, for any* $u, v \in V$, $u \lhd v$ *implies* $\lambda(u)\ D_{\widetilde{\Sigma}}\ \lambda(v)$.

The set of M^+-traces over $\widetilde{\Sigma}$ is denoted by $\mathbb{TR}^+(\widetilde{\Sigma})$, the set of M^--traces over $\widetilde{\Sigma}$ by $\mathbb{TR}^-(\widetilde{\Sigma})$. As usual, we often write $\mathbb{TR}^+$ (not to be confused with the product operation) or $\mathbb{TR}^-$ if $\widetilde{\Sigma}$ can be learned from the context. Observe that $\mathbb{TR}^+(\widetilde{\Sigma})$ might be seen as a subset of $\mathbb{DAG}(\widetilde{\Sigma}, 2^{Ag} \setminus \{\emptyset\})$. Moreover, $\mathbb{TR}^-(\widetilde{\Sigma}) \subseteq \mathbb{DAG}_{\rightrightarrows}(\widetilde{\Sigma})$, whereas, in general, $\mathbb{TR}^+(\widetilde{\Sigma}) \subseteq \mathbb{DAG}_{\rightrightarrows}(\widetilde{\Sigma}, 2^{Ag})$ does not hold. Remarkably, if $D_{\widetilde{\Sigma}} = \Sigma \times \Sigma$, then any trace (no matter if M^+or M^-) constitutes a totally ordered set and we even have $\mathbb{TR}^-(\widetilde{\Sigma}) \subseteq \mathbb{W}(\Sigma)$.

Whenever we leave open whether we deal with an M^+- or an M^--trace over $\widetilde{\Sigma}$, we write $(V, \prec, \lambda)$ to refer to either $(V, \{\lhd_\ell\}_{\ell \in 2^{Ag}}, \lambda)$ or $(V, \lhd, \lambda)$, respectively. Accordingly, $\prec'$ is actually meant to be either a collection of relations $\{\lhd'_\ell\}_{\ell \in 2^{Ag}}$ or a relation $\lhd'$.

Example 6.3. Recall that part (a) of Fig. 5.4 on page 48 depicts an M^+- trace over the distributed alphabet $(\{a,b,d\},\{a,b,e\},\{a,b\})$, whereas Fig. 5.4b shows an M^--trace. Moreover, Fig. 6.1a and Fig. 6.1b depict an M^+- and an M^--trace over $(\{a,b\},\{b,c\},\{c\})$, respectively (say, with $Ag = \{1,2,3\}$).

Usually, Mazurkiewicz traces are defined as labeled posets $(V,\leq,\lambda)$ [27]. More specifically, we call a Σ-labeled poset $(V,\leq,\lambda)$ a *poset-trace* over $\widetilde{\Sigma}$ if

- for any $u,v \in V$, if $\lambda(u)\ D_{\widetilde{\Sigma}}\ \lambda(v)$, then $u \leq v$ or $v \leq u$, and
- for any $u,v \in V$, $u \lessdot v$ implies $\lambda(u)\ D_{\widetilde{\Sigma}}\ \lambda(v)$.

But to treat all the structures relevant to this book in a common framework, a trace is given by its graphical representations. However, there is a one-to-one correspondence of poset-, M^+-, and M^--traces: for any $\alpha \in \{+,-\}$ and any poset-trace $\mathcal{P}$ over $\widetilde{\Sigma}$, there is a unique M^α-trace $\mathcal{T} = (V,\prec,\lambda) \in \mathbb{TR}^\alpha(\widetilde{\Sigma})$ (recall that $\prec$ is meant to be either $\{\lhd_\ell\}_{\ell\in 2^{Ag}}$ or $\lhd$) such that $(V,\leq,\lambda) = \mathcal{P}$. It is therefore justified to call $\mathcal{T}$ *the* M^α*-trace* of $\mathcal{P}$. Moreover, we can state that, for any given word $\underline{w} \in \mathbb{W}(\Sigma)$, there is exactly one M^α-trace $\mathcal{T}$ such that $\underline{w} \in \mathit{Lin}(\mathcal{T})$. For example, Fig. 6.1 depicts the only M^+-trace/M^--trace over $(\{a,b\},\{b,c\},\{c\})$ with linearization *cbacbb*. Thus, the traces from Fig. 6.1 constitute two different views of one and the same behavior.

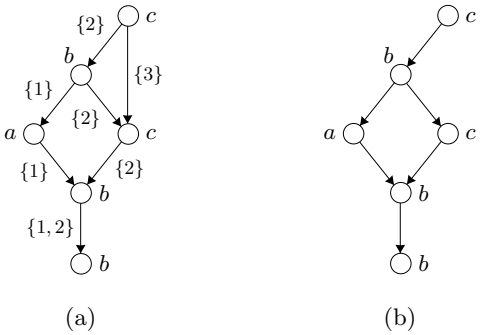


Fig. 6.1. An M^+-trace and an M^--trace over $(\{a,b\},\{b,c\},\{c\})$

6.2 Trace Languages

For this section, we fix $\alpha \in \{+,-\}$. For M^α-traces $\mathcal{T} = (V,\prec,\lambda)$ and $\mathcal{T}' = (V',\prec',\lambda')$ over $\tilde{\Sigma}$, let $\mathcal{T}\cdot\mathcal{T}'$ denote the *concatenation* of $\mathcal{T}$ and $\mathcal{T}'$, which is the M^α-trace of the poset-trace $(V'',\leq'',\lambda'')$ over $\widetilde{\Sigma}$ with $V'' = V \cup V'$, $\lambda'' = \lambda \cup \lambda'$, and $\leq'' = (\lhd \cup \lhd' \cup \{(u,v) \in V\times V' \mid (\lambda(u),\lambda'(v)) \in D_{\widetilde{\Sigma}}\})^*$. It is an easy task to show that trace concatenation is associative.

We denote by $\mathbf{1}_{\mathbb{TR}^\alpha}$ the empty M^α-trace, i.e., either $(\emptyset, \{\emptyset\}_{\ell \in 2^{Ag}}, \emptyset)$ or $(\emptyset, \emptyset, \emptyset)$. Then, $(\mathbb{TR}^\alpha, \cdot, \mathbf{1}_{\mathbb{TR}^\alpha})$ is a monoid, called the M^α-*trace monoid* over $\widetilde{\Sigma}$, which is mostly identified with $\mathbb{TR}^\alpha$. As usual, we assume in most definitions a trace to be nonempty. However, it will be clear how to incorporate $\mathbf{1}_{\mathbb{TR}^\alpha}$, too, which we do silently.

We have already implicitly defined $\mathcal{RAT}_{\mathbb{TR}^\alpha(\widetilde{\Sigma})}$ and $\mathcal{REC}_{\mathbb{TR}^\alpha(\widetilde{\Sigma})}$, the classes of rational and recognizable M^α-trace languages, respectively. Moreover, we have introduced the classes $\mathcal{MSO}(\Sigma, 2^{Ag})_{\mathbb{TR}^+(\widetilde{\Sigma})}$ and $\mathcal{MSO}(\Sigma)_{\mathbb{TR}^-(\widetilde{\Sigma})}$ of $\mathrm{MSO}(\Sigma, 2^{Ag})_{\mathbb{TR}^+(\widetilde{\Sigma})}$-definable M^+-trace languages and $\mathrm{MSO}(\Sigma)_{\mathbb{TR}^-(\widetilde{\Sigma})}$-definable M^--trace languages. In particular, the MSO formulas tailored to M^+-traces over $\widetilde{\Sigma}$ are built from the atomic entities

$$\lambda(x) = a \qquad x \lhd_\ell y \qquad x \in X \qquad x = y$$

and those tailored to M^--traces over $\widetilde{\Sigma}$ are built from the formulas

$$\lambda(x) = a \qquad x \lhd y \qquad x \in X \qquad x = y$$

(where $x, y \in \mathrm{Var}$, $a \in \Sigma$, $\ell \in 2^{Ag}$, and $X \in \mathrm{VAR}$). For their semantics, see the semantics of $\mathrm{MSO}(\Sigma, 2^{Ag})$ and $\mathrm{MSO}(\Sigma)$, respectively.

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

Theorem 6.4 ([79]).

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

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

Definition 6.5 (Regular Trace Language). *We call a set* $L \subseteq \mathbb{TR}^\alpha(\widetilde{\Sigma})$ regular *if* $Lin(L) \in \mathcal{REC}_{\mathbb{W}(\Sigma)}$*, i.e., if* $Lin(L)$ *is a regular word language over* Σ*.*

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

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 $\vdash_{\widetilde{\Sigma}}$. Namely, given a set $L \subseteq \mathbb{TR}^\alpha(\widetilde{\Sigma})$ and a trace $\mathcal{T} \in \mathbb{TR}^\alpha(\widetilde{\Sigma})$, we write $L \vdash_{\widetilde{\Sigma}} \mathcal{T}$ if the following holds:

$$\forall i \in Ag : \; \exists \mathcal{T}' \in L : \; \mathcal{T}' \restriction i = \mathcal{T} \restriction i$$

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

Definition 6.6 (Product Trace Language, cf. [88]). *We call a set of traces $L \subseteq \mathbb{TR}^\alpha$ a* weak product (trace) language *if, for any $\mathcal{T} \in \mathbb{TR}^\alpha$, $L \vdash_{\widetilde{\Sigma}} \mathcal{T}$ implies $\mathcal{T} \in L$. A set $L \subseteq \mathbb{TR}^\alpha$ is called a* product (trace) language *if it is the finite union of weak product trace languages.*

Let us denote by $\mathcal{P}^0_{\mathbb{TR}^\alpha}$ and $\mathcal{P}_{\mathbb{TR}^\alpha}$ the classes of weak product trace languages and product trace languages, respectively.

Example 6.7. Suppose $\widetilde{\Sigma} = (\{a, b\}, \{b, c\})$ and let L consist of all those traces $(V, \prec, \lambda) \in \mathbb{TR}^\alpha(\widetilde{\Sigma})$ such that there are $u, v \in V$ with $\lambda(u) = a$, $\lambda(v) = c$, $u \not\preceq v$, and $v \not\preceq u$. Then, though it is a regular trace language, L is not a weak product language (we will see below that it is not even a product language). Because if we suppose the trace $\mathcal{T}_1 \in L$ to be given by its projections $\mathcal{T}_1 \restriction 1 = ab$ and $\mathcal{T}_1 \restriction 2 = cb$, while $\mathcal{T}_2 \in L$ is given by $\mathcal{T}_2 \restriction 1 = ba$ and $\mathcal{T}_2 \restriction 2 = bc$, then we have both $\{\mathcal{T}_1, \mathcal{T}_2\} \vdash_{\widetilde{\Sigma}} abc \notin L$ (witnessed by $\mathcal{T}_1 \restriction 1$ and $\mathcal{T}_2 \restriction 2$) and $\{\mathcal{T}_1, \mathcal{T}_2\} \vdash_{\widetilde{\Sigma}} cba \notin L$ (witnessed by $\mathcal{T}_1 \restriction 2$ and $\mathcal{T}_2 \restriction 1$), which contradicts the definition of a weak product language. Thus, L is not contained in $\mathcal{P}^0_{\mathbb{TR}^\alpha(\widetilde{\Sigma})}$.

Let us bring together the concepts of regularity and product behavior: a trace language $L \subseteq \mathbb{TR}^\alpha$ is called a *weak regular product language* if both $L \in \mathcal{R}_{\mathbb{TR}^\alpha}$ and $L \in \mathcal{P}^0_{\mathbb{TR}^\alpha}$. Moreover, it is said to be a *regular product language* if it is the finite union of weak regular product languages. The corresponding language classes are denoted by $\mathcal{RP}^0_{\mathbb{TR}^\alpha}$ and, respectively, $\mathcal{RP}_{\mathbb{TR}^\alpha}$.

6.3 Asynchronous Automata

The distributed nature of traces asks for likewise distributed automata models, which preferably cover some of the classes proposed in the previous section. Let us start with asynchronous automata, the most general version of automata for traces that we consider, and let us fix $\alpha \in \{+, -\}$ for the moment.

Definition 6.8 (Asynchronous Automaton [97]). *An* asynchronous automaton *over $\widetilde{\Sigma}$ is a structure*

$$\mathcal{A} = ((S_i)_{i \in Ag}, (\Delta_a)_{a \in \Sigma}, \overline{s}^{in}, F)$$

where

- *for each $i \in Ag$, 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*

$$S_a := \{\overline{s} \in \prod_{i \in Ag} (S_i \cup \{*\}) \mid \text{ for any } i \in Ag,\ \overline{s}[i] = * \text{ iff } i \notin loc(a)\},$$

- $\overline{s}^{in} \in \prod_{i \in Ag} S_i$ *is the* global initial state*, and*

- $F \subseteq \prod_{i \in Ag} S_i$ *is the set of* global final states.

Instead of $(\overline{s}, \overline{s}') \in \Delta_a$, we also write more clearly $\overline{s} \xrightarrow{a} \overline{s}'$. We call $\mathcal{A}$ *deterministic* if, for any $a \in \Sigma$ and any $\overline{s} \in S_a$, there is at most one $\overline{s}' \in S_a$ such that $\overline{s} \xrightarrow{a} \overline{s}'$. We will see that, for any M^α-trace $\mathcal{T}$, no matter which mode α we choose, a deterministic asynchronous automaton will allow at most one run on $\mathcal{T}$.

Example 6.9. An asynchronous automaton over $(\{a, b\}, \{b, c\})$ that is not deterministic is illustrated in Fig. 6.2 where b-synchronizing transitions are joined by dashed lines. Thus, its transitions are those reflected by Table 6.1.

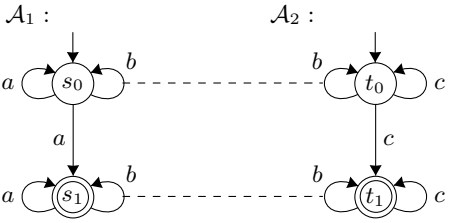


Fig. 6.2. An asynchronous automaton over $(\{a, b\}, \{b, c\})$

Table 6.1. The transitions of the asynchronous automaton from Fig. 6.2

$(s_0, *) \xrightarrow{a} (s_0, *)$
$(s_0, *) \xrightarrow{a} (s_1, *)$
$(s_1, *) \xrightarrow{a} (s_1, *)$
$(s_0, t_0) \xrightarrow{b} (s_0, t_0)$
$(s_1, t_1) \xrightarrow{b} (s_1, t_1)$
$(*, t_0) \xrightarrow{c} (*, t_0)$
$(*, t_0) \xrightarrow{c} (*, t_1)$
$(*, t_1) \xrightarrow{c} (*, t_1)$

Let $\mathcal{A} = ((S_i)_{i \in Ag}, (\Delta_a)_{a \in \Sigma}, \overline{s}^{in}, F)$ be an asynchronous automaton and let $\mathcal{T} = (V, \prec, \lambda) \in \mathbb{TR}^\alpha$ be a trace. A mapping $\rho : V \to \bigcup_{a \in \Sigma} S_a$ satisfying $\rho(u) \in S_{\lambda(u)}$ for each $u \in V$ is called a *run* of $\mathcal{A}$ on $\mathcal{T}$ if, for any $u \in V$, we have

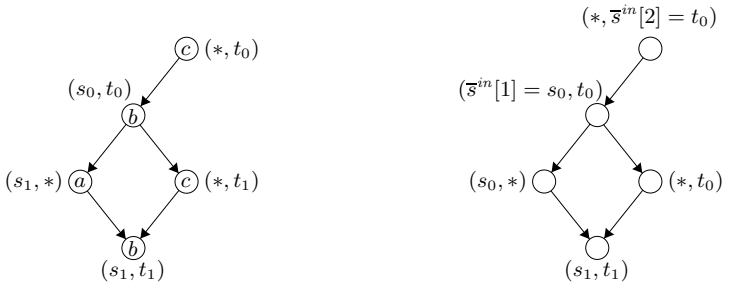
$$\overline{s} \xrightarrow{\lambda(u)} \rho(u)$$

where $\overline{s} \in S_{\lambda(u)}$ is determined by

$$\overline{s}[i] = source_{(\mathcal{T},\rho)}^{(\overline{s}^{in})_{i\in Ag}}(u)[i][i]$$

for any $i \in loc(\lambda(u))$. We call ρ *accepting* if $\overline{f} \in F$ with $\overline{f}[i] = final_{(\mathcal{T},\rho)}^{(\overline{s}^{in})_{i\in Ag}}[i][i]$ for any $i \in Ag$. The M^α-*language* of $\mathcal{A}$, $\{\mathcal{T} \in \mathbb{TR}^\alpha \mid$ there is an accepting run of $\mathcal{A}$ on $\mathcal{T}\}$, is denoted by $L^\alpha(\mathcal{A})$.

Example 6.10. An accepting run ρ of the asynchronous automaton $\mathcal{A}$ from Fig. 6.2 on some M^--trace $\mathcal{T}$ over $(\{a,b\},\{b,c\})$ is illustrated in part (a) of Fig. 6.3, which also reflects the sources of a transition (b). One verifies that the M^α-language of $\mathcal{A}$ is the set of traces $(V,\prec,\lambda) \in \mathbb{TR}^\alpha((\{a,b\},\{b,c\}))$ such that there is $u,v \in V$ with $\lambda(u) = a$, $\lambda(v) = c$, $u \not\leq v$, and $v \not\leq u$. From Example 6.7, we know that $L^\alpha(\mathcal{A}) \notin \mathcal{P}^0_{\mathbb{TR}^\alpha((\{a,b\},\{b,c\}))}$.



(a) a run ρ on $\mathcal{T}$ (b) the sources of a transition

Fig. 6.3. An accepting run of an asynchronous automaton

Example 6.11. Consider two processes, 1 and 2. Process 1 has access to variables `x` and `y`, process 2 can write the variables `x` and `z`. Thus, 1 and 2 share a common read and write domain in terms of variable `x`. We suppose that, for some $n \in \mathbb{N}$, `x`, `y`, and `z` range over $\{0,\ldots,n\}$ and that they are each initialized to 0. Both processes can increase the value of `x` by 1. Moreover, processes 1 and 2 count up their exclusive variables `y` and `z`, respectively. Accordingly, we deal with a distributed alphabet $\widetilde{\Sigma}$ given by $\Sigma_1 = \{\texttt{x:=x+1}, \texttt{y:=y+1}\}$ and $\Sigma_2 = \{\texttt{x:=x+1}, \texttt{z:=z+1}\}$. We would like to implement a system in which the value of `x` can be increased only if the values of `y` and `z` coincide. Moreover, we suppose the system to accept a computation only if `x` is evaluated to n. Such a system might be represented by the asynchronous automaton $\mathcal{A}_n = ((S_i)_{i\in\{1,2\}}, (\Delta_a)_{a\in\Sigma}, \overline{s}^{in}, F)$ over $\widetilde{\Sigma}$ where

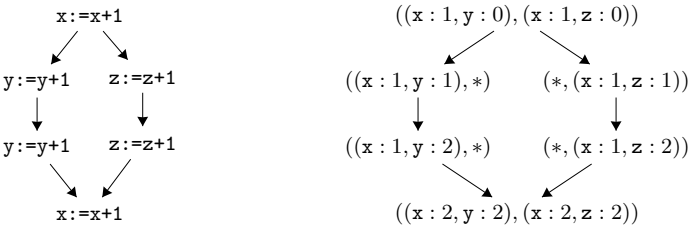
- $S_1 = \{0,\ldots,n\}^{\{\texttt{x},\texttt{y}\}}$ and $S_2 = \{0,\ldots,n\}^{\{\texttt{x},\texttt{z}\}}$, i.e., a local state represents a variable evaluation mapping a variable to a value not greater than n,

- $\overline{s}^{in}[i][\mathtt{x}] = \overline{s}^{in}[i][\mathtt{x}] = 0$ for $i = 1, 2$ and $\overline{s}^{in}[1][\mathtt{y}] = \overline{s}^{in}[2][\mathtt{z}] = 0$, and
- $F = \{\overline{s} \in S_1 \times S_2 \mid \overline{s}[1][\mathtt{x}] = \overline{s}[2][\mathtt{x}] = n\}$.

Moreover, the transitions of $\mathcal{A}_n$ shall be given as follows:

- for any $s, s' \in S_1$, $(s, *) \xrightarrow{\mathtt{y:=y+1}} (s', *)$ iff both $s[\mathtt{y}]+1 = s'[\mathtt{y}]$ and $s[\mathtt{x}] = s'[\mathtt{x}]$,
- for any $s, s' \in S_2$, $(*, s) \xrightarrow{\mathtt{z:=z+1}} (*, s')$ iff both $s[\mathtt{z}]+1 = s'[\mathtt{z}]$ and $s[\mathtt{x}] = s'[\mathtt{x}]$, and
- for any $\overline{s}, \overline{s}' \in S_{\mathtt{x:=x+1}} = S_1 \times S_2$, we have the transition $\overline{s} \xrightarrow{\mathtt{x:=x+1}} \overline{s}'$ iff
  - $\overline{s}[1][\mathtt{y}] = \overline{s}[2][\mathtt{z}] = \overline{s}'[1][\mathtt{y}] = \overline{s}'[2][\mathtt{z}]$ and
  - $\overline{s}[1][\mathtt{x}] + 1 = \overline{s}[2][\mathtt{x}] + 1 = \overline{s}'[1][\mathtt{x}] = \overline{s}'[2][\mathtt{x}]$.

Figure 6.4 depicts an M$^-$-trace from $L^-(\mathcal{A}_2)$ (i.e., with $n = 2$) and its run.



(a) an M$^-$-trace (b) an accepting run of $\mathcal{A}_2$

Fig. 6.4. A computation of $\mathcal{A}_2$ from Example 6.11

By $\mathcal{AA}^{\alpha}(\widetilde{\Sigma})$ (or simply $\mathcal{AA}^{\alpha}$), we denote $\{L^{\alpha}(\mathcal{A}) \mid \mathcal{A}$ is an asynchronous automaton over $\widetilde{\Sigma}\}$. The class det-$\mathcal{AA}^{\alpha}(\widetilde{\Sigma})$ is self-explanatory.

We may describe the behavior of an asynchronous automaton $\mathcal{A} = ((S_i)_{i \in Ag}, (\Delta_a)_{a \in \Sigma}, \overline{s}^{in}, F)$ equivalently in a rather operational (and maybe more intuitive) manner. This view is based on the *global transition relation* $\Longrightarrow_{\mathcal{A}}$ of $\mathcal{A}$, describing its global step-by-step behavior. Accordingly, $\Longrightarrow_{\mathcal{A}}$ is a subset of $S_{\mathcal{A}} \times \Sigma \times S_{\mathcal{A}}$ where $S_{\mathcal{A}} := \prod_{i \in Ag} S_i$ is the set of *global states* of $\mathcal{A}$. Intuitively, $\mathcal{A}$, being in some global state $\overline{s}$, can execute a letter a if, according to Δ_a, a is enabled in the collection of components from $loc(a)$. More precisely, we define $(\overline{s}, a, \overline{s}') \in S_{\mathcal{A}} \times \Sigma \times S_{\mathcal{A}}$ to be contained in $\Longrightarrow_{\mathcal{A}}$ if there is a transition $(\overline{t}, a, \overline{t}') \in \Delta_a$ such that

- for any $i \in loc(a)$, we have both $\overline{s}[i] = \overline{t}[i]$ and $\overline{s}'[i] = \overline{t}'[i]$, and
- for any $i \notin loc(a)$, we have $\overline{s}[i] = \overline{s}'[i]$.

Then, $\mathcal{A}$ defines in a natural manner a word language $L_{word}(\mathcal{A}) \subseteq \mathbb{W}(\Sigma)$, which is set to be $L((S_{\mathcal{A}}, \Longrightarrow_{\mathcal{A}}, \overline{s}^{in}, F))$, i.e., the language of the finite automaton $(S_{\mathcal{A}}, \Longrightarrow_{\mathcal{A}}, \overline{s}^{in}, F)$ over Σ.

Example 6.12. The finite automaton of the asynchronous automaton from Fig. 6.2 is depicted in Fig. 6.5. Observe that its language L is the one of the rational expression $(a+b+c)^*(ac+ca)(a+b+c)^*$ of $\mathbb{W}(\{a,b,c\})$. Moreover, L is precisely $Lin(L(\beta))$ if β is the rational expression $(a+b+c)^*ac(a+b+c)^*$ of $\mathbb{TR}^\alpha((\{a,b\},\{b,c\}))$ (no matter if $\alpha = +$ or $\alpha = -$) where, actually, any letter stands for an M^α-trace. In fact, we have the following correspondence, which shows $\mathcal{AA}^\alpha \subseteq \mathcal{R}_{\mathbb{TR}^\alpha}$:

Lemma 6.13. *For any $\alpha \in \{+,-\}$ and any asynchronous automaton $\mathcal{A}$ over $\widetilde{\Sigma}$, we have $L_{word}(\mathcal{A}) = Lin(L^\alpha(\mathcal{A}))$.*

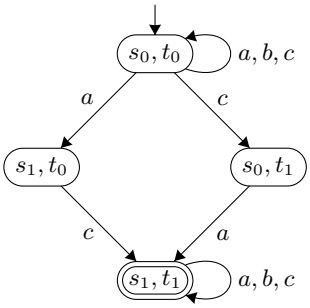


Fig. 6.5. The finite automaton of the asynchronous automaton from Fig. 6.2

Exercise 6.14. Let $\mathcal{A}$ be an asynchronous automaton over $\widetilde{\Sigma}$. Show that, for any $\underline{w}_1, \underline{w}_2 \in \mathbb{W}(\Sigma)$ and $a, b \in \Sigma$ with $a\ I_{\widetilde{\Sigma}}\ b$, we have $\underline{w}_1 ab \underline{w}_2 \in L_{word}(\mathcal{A})$ iff $\underline{w}_1 ba \underline{w}_2 \in L_{word}(\mathcal{A})$.

Exercise 6.15. Show Lemma 6.13.

Exercise 6.16. Build the finite automaton of $\mathcal{A}_3$ from Example 6.11.

6.4 Asynchronous Automata vs. ACAs and EMSO Logic

When looking for a monadic second-order logic characterizing asynchronous automata, we profit from the results achieved in Chap. 5, where ACAs were characterized relative to traces in terms of EMSO logic. There exist easy effective translations from ACAs to asynchronous automata and vice versa so that a logical characterization of asynchronous automata follows immediately.

Theorem 6.17.

$$\mathcal{AA}^+ \;=\; \mathcal{ACA}_{\mathbb{TR}^+}$$

Proof. "$\supseteq$": Suppose $\mathcal{A} = (Q, \Delta, T, F)$ is an ACA over $(\widetilde{\Sigma}, 2^{Ag})$. Without loss of generality, we assume $\mathcal{A}$ to be state separated (cf. Exercise 5.23). To simulate a run of $\mathcal{A}$ by an asynchronous automaton, the latter employs $Q \uplus \{\imath\}$ as local state spaces. When reading, say, a b-labeled node u, it moreover collects the states of the predecessors of u, selects from each state a component belonging to $loc(b)$, and compares such information with the transitions of $\mathcal{A}$. So consider the asynchronous automaton $\mathcal{A}' = ((S_i)_{i \in Ag}, (\Delta_a)_{a \in \Sigma}, \overline{s}^{in}, F')$ over $\widetilde{\Sigma}$, which is given by

- $S_i = Q \uplus \{\imath\}$ for any $i \in Ag$,
- for $b \in \Sigma$ and $\overline{s}, \overline{s}' \in S_b$, we have $\overline{s} \xrightarrow{b} \overline{s}'$ iff there are $t = (\overline{q}, b, q) \in \Delta$, $\mathcal{T} = (V, \{\lhd_\ell\}_{\ell \in 2^{Ag}}, \lambda) \in \langle \mathbb{TR}^+(\widetilde{\Sigma}), Q \rangle$, and $u \in V$ such that
  - $trans_{\mathcal{T}}(u) = t$,
  - $\overline{s} = source_{\mathcal{T}}^{(\imath)_{i \in Ag}}(u)$, and
  - for any $i \in loc(b)$, we have $\overline{s}'[i] = q$,
- $\overline{s}^{in} = (\imath)_{i \in Ag}$, and
- $F' = F$.

In fact, we easily verify that $L^+(\mathcal{A}') = L_{\mathbb{TR}^+(\widetilde{\Sigma})}(\mathcal{A})$.

"$\subseteq$": Suppose $\mathcal{A} = ((S_i)_{i \in Ag}, (\Delta_a)_{a \in \Sigma}, \overline{s}^{in}, F)$ is an asynchronous automaton over $\widetilde{\Sigma}$. We construct an ACA $\mathcal{A}' = (Q, \Delta, T, F')$ over $(\widetilde{\Sigma}, 2^{Ag})$ with $L_{\mathbb{TR}^+(\widetilde{\Sigma})}(\mathcal{A}') = L^+(\mathcal{A})$, which, in a run, assigns to an a-labeled node a state from S_a and checks by means of the transitions from Δ if the assignment corresponds to an accepting run of $\mathcal{A}$. Accordingly, we specify $\mathcal{A}'$ as follows:

- $Q = \bigcup_{a \in \Sigma} S_a$,
- $\Delta = \{trans_{(\mathcal{T}, \rho)}(u) \mid \mathcal{T} = (V, \{\lhd_\ell\}_{\ell \in 2^{Ag}}, \lambda) \in \mathbb{TR}^+(\widetilde{\Sigma})$, ρ is a run of $\mathcal{A}$ on $\mathcal{T}$, and $u \in V\}$,
- $T(a, q) = \emptyset$ for any $(a, q) \in \Sigma \times Q$, and
- $F' = \{final_{(\mathcal{T}, \rho)}^{(\imath)_{i \in Ag}} \mid \mathcal{T} \in \mathbb{TR}^+(\widetilde{\Sigma})$ and ρ is an accepting run of $\mathcal{A}$ on $\mathcal{T}\}$. $\square$

As the transformations from the proof of Theorem 6.17 are more or less immediate, we can justifiably state that

> *Asynchronous automata over $\widetilde{\Sigma}$ are*
> *Asynchronous cellular automata relative to $\mathbb{TR}^+(\widetilde{\Sigma})$.*

Exercise 6.18. Apply the construction from the proof of Theorem 6.17 to the asynchronous automaton that is depicted in Fig. 6.2.

Exercise 6.19. Argue that

> *Finite automata over Σ are*
> *Asynchronous cellular automata relative to $\mathbb{W}(\Sigma)$.*

Theorem 6.20.

$$\mathcal{AA}^- \;=\; \mathcal{ACA}_{\mathbb{TR}^-}$$

The proof Theorem 6.20 is more involved and cannot be done immediately. The problem here is that an ACA over M^--traces might not have immediate access to the latest event of any agent. There exist, however, tools such as *asynchronous mappings* that, in particular, allow us to transform an asynchronous automaton into an equivalent ACA relative to M^--traces [24, 27, 29]. With Theorem 5.45 and Corollary 5.32, we obtain the following:

Corollary 6.21.

(a) $\mathcal{AA}^+(\widetilde{\Sigma}) \;=\; \mathcal{EMSO}(\Sigma, 2^{Ag})_{\mathbb{TR}^+(\widetilde{\Sigma})}$

(b) $\mathcal{AA}^-(\widetilde{\Sigma}) \;=\; \mathcal{EMSO}(\Sigma)_{\mathbb{TR}^-(\widetilde{\Sigma})}$

Observe that, as the graph of an M^--trace is a Hasse diagram, we have $\mathcal{EMSO}(\Sigma)_{\mathbb{TR}^-(\widetilde{\Sigma})} = \mathcal{EMSO}(\Sigma)[\lessdot]_{\mathbb{TR}^-(\widetilde{\Sigma})}$.

We study the expressiveness of asynchronous automata further on and, for the rest of this section, let $\alpha \in \{+, -\}$ be arbitrary. The following is the celebrated theorem of Zielonka and bridges the gap between asynchronous automata and the algebraically motivated notions of recognizability and regularity. It states that one can (effectively) synthesize a distributed system in terms of an asynchronous automaton given a regular set of words that arises from a trace language.

Theorem 6.22 (Zielonka [97]).

$$\mathcal{REC}_{\mathbb{TR}^\alpha} \;=\; \mathcal{R}_{\mathbb{TR}^\alpha} \;=\; \mathcal{AA}^\alpha \;=\; \text{det-}\mathcal{AA}^\alpha$$

Theorem 4.10, 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 their graph structures or the underlying partial-order relation.

Theorem 6.23 (cf. [31, 90]).

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

Let us compare asynchronous automata with graph acceptors relative to traces.

Corollary 6.24.

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

Proof. Analogously to the word case, the second equality follows from Theorem 3.24 and Corollary 6.21. 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 [92]. □

6.5 Product Automata

Let us turn to (weak) regular product languages and try to find a suitable automata model, which is weaker than the asynchronous one and generates a regular product language in a natural manner. Synchronizing transitions turn out to be essential to recognizing the language of the automaton from Fig. 6.2 on page 81. 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. Let us again fix $\alpha \in \{+, -\}$.

Definition 6.25 (Product Automaton). *A* product automaton *over* $\widetilde{\Sigma}$ *is a structure* $\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \overline{s}^{in}, F)$ *such that*

- *for each* $i \in Ag$, $\mathcal{A}_i$ *is a pair* (S_i, Δ_i) *where*
  - 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*,*
- $\overline{s}^{in} \in \prod_{i \in Ag} S_i$ *is the* global initial state*, and*
- $F \subseteq \prod_{i \in Ag} S_i$ *is the set of* global final states*.*

A simple product automaton over $(\{a, b\}, \{b, c\})$ (say, $Ag = \{1, 2\}$) is illustrated in Fig. 6.6 where global final states are depicted by dashed lines.

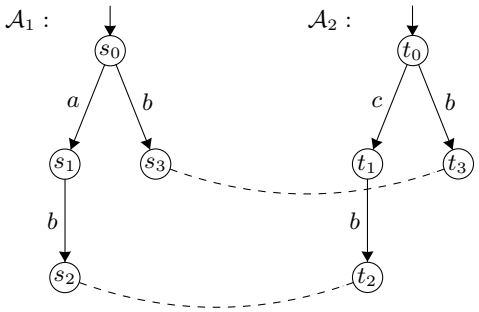


Fig. 6.6. A product automaton over $(\{a, b\}, \{b, c\})$

The behavior of a product automaton is quite similar to that of an asynchronous automaton but more autonomous. So let $\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be a product automaton over $\tilde{\Sigma}$ (again, set S_a to be $\{\overline{s} \in \prod_{i \in Ag}(S_i \cup \{*\}) \mid$ for any $i \in Ag$, $\overline{s}[i] = *$ iff $i \notin loc(a)\}$) and suppose $\mathcal{T} = (V, \prec, \lambda)$ to be an M^{α}-trace over $\widetilde{\Sigma}$. A *run* of $\mathcal{A}$ on $\mathcal{T}$ is a mapping $\rho : V \to \bigcup_{a \in \Sigma} S_a$ with $\rho(u) \in S_{\lambda(u)}$ (for each $u \in V$) such that, for any $u \in V$ and any $i \in loc(\lambda(u))$,

$$(source^{(\overline{s}^{in})_{i\in Ag}}_{(\mathcal{T},\rho)}(u)[i][i], \lambda(u), \rho(u)[i]) \in \Delta_i$$

The acceptance condition carries over from asynchronous automata. The M^α-*language* of $\mathcal{A}$, $\{\mathcal{T} \in \mathbb{TR}^\alpha \mid$ there is an accepting run of $\mathcal{A}$ on $\mathcal{T}\}$, is denoted by $L^\alpha(\mathcal{A})$. For example, the M^α-language of the product automaton from Fig. 6.6 is the one of the rational expression $b + acb$ of $\mathbb{TR}^\alpha((\{a, b\}, \{b, c\}))$. By $\mathcal{PA}^\alpha(\widetilde{\Sigma})$ (or just $\mathcal{PA}^\alpha$), we moreover denote the set of M^α-trace languages that is determined by the class of product automata over $\widetilde{\Sigma}$.

As already done for asynchronous automata, we may describe the behavior of a product automaton $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, by means of a *global transition relation* $\Longrightarrow_{\mathcal{A}}$. Again, $\Longrightarrow_{\mathcal{A}}$ is a subset of $S_{\mathcal{A}} \times \Sigma \times S_{\mathcal{A}}$ where $S_{\mathcal{A}} := \prod_{i\in Ag} S_i$ is the set of *global states* of $\mathcal{A}$. Moreover, $(\overline{s}, a, \overline{s}') \in S_{\mathcal{A}} \times \Sigma \times S_{\mathcal{A}}$ is contained in $\Longrightarrow_{\mathcal{A}}$ if

- for any $i \in loc(a)$, we have $(\overline{s}[i], a, \overline{s}'[i]) \in \Delta_i$, and
- for any $i \notin loc(a)$, we have $\overline{s}[i] = \overline{s}'[i]$.

Canonically, $\mathcal{A}$ determines a word language $L_{word}(\mathcal{A}) \subseteq \mathbb{W}(\Sigma)$, which is the language of the finite automaton $(S_{\mathcal{A}}, \Longrightarrow_{\mathcal{A}}, \overline{s}^{in}, F)$. Once we have shown Lemma 6.13, the following lemma is an easy consequence thereof.

Lemma 6.26. *For any $\alpha \in \{+, -\}$ and any product automaton $\mathcal{A}$ over $\widetilde{\Sigma}$, we have $L_{word}(\mathcal{A}) = Lin(L^\alpha(\mathcal{A}))$.*

The finite automaton of the product automaton $\mathcal{A}$ from Fig. 6.6 is depicted in Fig. 6.7. Its language is the finite set $\{b, acb, cab\}$, which, in fact, is precisely $Lin(L^\alpha(\mathcal{A}))$.

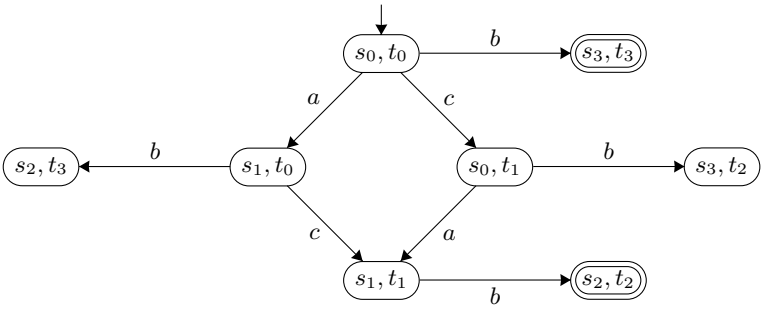


Fig. 6.7. The finite automaton of the product automaton from Fig. 6.6

Executing an action a, an asynchronous automaton has a *global* view on the current state of each agent involved in a. It can therefore globally decide to take a transition. In contrast, a product automaton has a *local* view on the collection of agents involved in a, which, in general, is strictly less expressive.

Lemma 6.27. *In general,*

$$\mathcal{PA}^{\alpha} \subsetneq \mathcal{AA}^{\alpha}$$

Proof. Suppose $\widetilde{\Sigma} = (\{a, b\}, \{b, c\})$ (say, $Ag = \{1, 2\}$) and suppose that L consists of those traces $(V, \prec, \lambda) \in \mathbb{TR}^{\alpha}(\widetilde{\Sigma})$ such that there is $u, v \in V$ with $\lambda(u) = a$, $\lambda(v) = c$, $u \not\leq v$, and $v \not\leq u$. Recall that L is recognized by the asynchronous automaton illustrated in Fig. 6.2 on page 81. Now suppose there is a product automaton $\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \overline{s}^{in}, F)$ over $\widetilde{\Sigma}$ with $L^{\alpha}(\mathcal{A}) = L$. For $m, n \in \mathbb{N}$, consider the trace $\mathcal{T}(m, n) \in \mathbb{TR}^{\alpha}(\widetilde{\Sigma})$, which is given by its linearization $b^m a c b^n$. Let us adopt the view of the global transition relation of $\mathcal{A}$. If m and n are sufficiently large, $\mathcal{A}_1$, in a successful run of $\mathcal{A}$ on $\mathcal{T}(m, n)$, goes through a cycle, say, of length k (≥ 1), to read the first m b's of the word $b^m a b^n$. Similarly, $\mathcal{A}_2$, in the same run, goes through a cycle, say of length l (≥ 1), to read the second n b's of $b^m c b^n$. But then, we can easily construct an accepting run of $\mathcal{A}$ on $b^m c b^{k \cdot l} a b^n \in Lin(\mathbb{TR}^{\alpha}(\widetilde{\Sigma}) \setminus L)$, which contradicts the premise. □

However, in Sect. 6.2, we identified a class of languages that precisely corresponds to $\mathcal{PA}^{\alpha}$.

Theorem 6.28 ([88]).

$$\mathcal{RP}_{\mathbb{TR}^{\alpha}} = \mathcal{PA}^{\alpha}$$

Note that $\mathcal{RP}^{0}_{\mathbb{TR}^{\alpha}}$ 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 MSCs.

Exercise 6.29.

(a) Show that $\mathcal{RP}^{0}_{\mathbb{TR}^{\alpha}}$ corresponds to the class of product automata where the set of global final states is a cartesian product.
(b) Show Theorem 6.28.

6.6 Summary

Table 6.2 exhibits important closure properties of asynchronous and product automata, which do not depend on whether the automata run on M^{+}-traces or M^{-}-traces (or even poset traces). Consequently, the concrete representation of a trace and the power of related atomic formulas does not influence the power of automata and logics. Missing proofs concerning the results for product automata can be found in [88]. Note that the definition of a deterministic product automaton has been omitted. It is, however, straightforward to call a product automaton deterministic whose local (finite) automata are.

Table 6.2. Closure and expressiveness properties of asynchronous automata

	$\cup$	$\cap$	$\bar{\cdot}$	det	EMSO	MSO	Empt.
$\mathcal{AA}^+$	✓	✓	✓	$=$	$=$	$=$	✓
$\mathcal{AA}^-$	✓	✓	✓	$=$	$=$	$=$	✓
$\mathcal{PA}^+$	✓	✓	✓	$=$	$\supsetneq$	$\supsetneq$	✓
$\mathcal{PA}^-$	✓	✓	✓	$=$	$\supsetneq$	$\supsetneq$	✓

6.7 Bibliographic Notes

A comprehensive overview of the theory of Mazurkiewicz traces and related automata models is provided by [27], which covers many important research lines of trace theory, including an algebraic view thereof. Recognizability of trace languages is the subject of [79]. In [96], recognizability and definability in terms of MSO logic are considered in terms of traces, MSCs, graphs, and infinite structures. Comparisons of the models of ACAs and asynchronous automata relative to traces can be found in [28] and [82]. Product automata and product trace languages are studied in [88].

7

Message Sequence Charts

As discussed in detail in Chap. 5, ACATs are capable of modeling systems of several components that communicate via message exchange through fifo channels. Resuming that study, we now model those systems and their channels explicitly. Afterwards, an embedding of message-passing systems into ACATs allows the transfer of several results into this more specific setting. Remember that the behavior of those systems can be described by a collection of *message sequence charts* (MSCs). In the following, let us recall some basic definitions that have already been settled in Example 5.12. However, defining MSCs, we will now come from the more general notion of *partial* MSCs, which, in contrast to MSCs, admit events to be unmatched.

7.1 Message Sequence Charts

As usual, let Ag be a finite set of at least two agents. Towards defining a distributed alphabet that is tailored to channel systems, we first let

$$\Gamma_i^s(Ag) := \{i!j \mid j \in Ag \setminus \{i\}\}$$
$$\Gamma_i^r(Ag) := \{i?j \mid j \in Ag \setminus \{i\}\}$$

denote the sets of *send* and, respectively, *receive actions* that are available to agent $i \in Ag$. Accordingly, $\Gamma_i(Ag) := \Gamma_i^s(Ag) \cup \Gamma_i^r(Ag)$ contains any action that i may execute, $\Gamma^s(Ag) := \bigcup_{i \in Ag} \Gamma_i^s(Ag)$ is the set of send, $\Gamma^r(Ag) := \bigcup_{i \in Ag} \Gamma_i^r(Ag)$ is the set of receive, and $\Gamma(Ag) := \Gamma^s(Ag) \cup \Gamma^r(Ag)$ is the set of all the actions. Here, the symbols $i!j$ and $j?i$ are to be read as "i sends a message to j" and "j receives a message from i", respectively. They are related in the sense that they will label communicating events of an MSC, which are joined by a message arrow in their graphical representation. We therefore set $Com(Ag) := \{(i!j, j?i) \mid (i,j) \in Ch(Ag)\}$. Recall that an action $i\theta j$ ($\theta \in \{!,?\}$) is performed by agent i, which is indicated by $Ag(i\theta j) = i$.

In a natural manner, a message-passing system over Ag is based on the distributed alphabet $\widetilde{\Gamma}(Ag) := (\Gamma_i(Ag))_{i \in Ag}$. Henceforth, we mostly take the liberty of omitting the reference to Ag and just write Γ^s, Γ^r, Γ, Com, and $\widetilde{\Gamma}$, for example.

MSCs will be defined stepwise, starting with *partial MSCs*, where some events may lack a suitable communication partner. The latter view is concretized towards *lossy MSCs*, which allow at most unmatched send events, whereas, finally, a *basic MSC* describes complete behavior without any open events.

Definition 7.1 (Partial Message Sequence Chart). *A* partial message sequence chart *(over Ag) is a $\widetilde{\Gamma}$-dag $(V, \lhd, \lambda)$ such that*

- *for any $i \in Ag$, $\lhd_i = \lessdot_i$, i.e., $\lhd_i$ is the covering relation of $\leq_i$, and*
- *for any $(u, v) \in \lhd_{\mathrm{c}}$, $(\lambda(u), \lambda(v)) \in Com$.*

Unlike the general case of dags over distributed alphabets, the first condition from Definition 7.1 guarantees that the class of partial MSCs is locally covering. Moreover, the definition of a $\widetilde{\Gamma}$-dag makes sure that completed message transfers in a partial MSC, which correspond to an edge whose nodes are labeled with communicating actions, are processed along a fifo architecture. However, there might still be unmatched events. So suppose $\mathcal{M} = (V, \lhd, \lambda)$ is a partial MSC over Ag. We identify events from V that either have no communication partner yet or just lack a link to an existing partner and let V_{um} denote the set $\{u \in V \mid \text{there is no } v \in V \text{ such that } u \lhd_{\mathrm{c}} v \text{ or } v \lhd_{\mathrm{c}} u\}$ of *unmatched events* of $\mathcal{M}$. Given $u \in V$, $Ag(u)$ will subsequently serve as a shorthand for $Ag(\lambda(u))$. Moreover, we may write $Ag(\mathcal{M})$ instead of $\{Ag(u) \mid u \in V\}$.

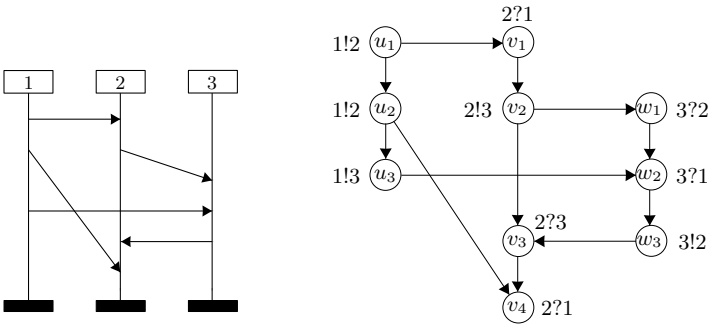
Definition 7.2 (Lossy Message Sequence Chart). *A* lossy message sequence chart *is a partial MSC $(V, \lhd, \lambda)$ such that, for any $u \in V_{\mathrm{um}}$, $\lambda(u) \in \Gamma^s$.*

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

Definition 7.3 (Message Sequence Chart). *A* message sequence chart *is a partial MSC $(V, \lhd, \lambda)$ such that we have $V_{\mathrm{um}} = \emptyset$.*

Example 7.4. An MSC $(V, \lhd, \lambda)$ over $\{1, 2, 3\}$ is depicted in Fig. 7.1b. However, to illustrate an MSC, we mostly represent it by a diagram such as shown in Fig. 7.1a, 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 Hasse diagrams, i.e., to graphs from $\mathbb{DAG}_H(\widetilde{\Gamma})$, as the edge representing the second message from agent 1 to agent 2 is already implicitly present. Observe that we have $V_1 = \{u_1, u_2, u_3\}$, $u_1 \lhd_1 u_2 \lhd_1 u_3$, $u_2 \lhd_{\mathrm{c}} v_4$ and $u_2 \lhd_{(1,2)} v_4$, $w_3 \lhd_{\mathrm{c}} v_3$ and $w_3 \lhd_{(3,2)} v_3$, $v_1 \leq_2 v_4$, $u_3 \leq v_3$, $u_3 \lessdot w_2$, and $u_2 \not\leq v_4$.

Example 7.5. Figure 7.2 shows a partial MSC that is not lossy (a), a lossy MSC that is not an MSC (b), and an MSC (c). If, in general, we deal with partial MSCs, we may indicate unmatched events as illustrated by Fig. 7.3 reflecting the graphs from Fig. 7.2. Hereby, the labeling of an unmatched event indicates the desired communication partner.



(a) (b)

Fig. 7.1. Two different views of an MSC

The set of partial MSCs over Ag is denoted by $\mathbb{PMSC}(Ag)$, the set of lossy MSCs by $\mathbb{LMSC}(Ag)$, and the set of MSCs by $\mathbb{MSC}(Ag)$. The members of $\mathbb{MSC}(Ag)$ are often called *basic* MSCs. An *MSC language* is a set of basic MSCs. When Ag is clear from the context, a corresponding reference may be omitted. Obviously, we have $\mathbb{MSC} \subseteq \mathbb{LMSC} \subseteq \mathbb{PMSC} \subseteq \mathbb{DAG}_{\rightrightarrows}(\widetilde{\Gamma})$.

To be able to apply the theory of graph acceptors and, in particular, Theorem 3.24, it is important to observe that $\mathbb{MSC}$, $\mathbb{LMSC}$, and $\mathbb{PMSC}$ have bounded degree. More precisely, their degree is bounded by 3. To verify this, observe that any event of a partial MSC has at most one communication partner as well as at most two direct neighbors on its process line.

Given a partial MSC $\mathcal{M} = (V, \lhd, \lambda) \in \mathbb{PMSC}$, we might argue that some pairs of unmatched events of $\mathcal{M}$ can be combined towards a complete message. Following this idea, we define a relation $\widehat{\lhd_{\mathrm{c}}} \subseteq V \times V$, which accordingly relates events from V_{um}. Intuitively, they are queued into a fifo channel as far as possible. For $u, v \in V$, we formally write $u \mathrel{\widehat{\lhd_{\mathrm{c}}}} v$ if $\{u, v\} \subseteq V_{\mathrm{um}}$, $(\lambda(u), \lambda(v)) \in \mathit{Com}$, and $|\mathcal{M} \Downarrow u|_{\lambda(u)} = |\mathcal{M} \Downarrow v|_{\lambda(v)}$. If $(V, \lhd \cup \widehat{\lhd_{\mathrm{c}}}, \lambda)$ is a partial MSC, then it can be understood as the adjustment of $\mathcal{M}$ along the fifo architecture. In that case, we say that $\mathcal{M}$ *represents* $(V, \lhd \cup \widehat{\lhd_{\mathrm{c}}}, \lambda)$, written $\mathcal{M} \preceq (V, \lhd \cup \widehat{\lhd_{\mathrm{c}}}, \lambda)$. For example, the partial MSC from Fig. 7.3a represents the one from Fig. 7.3b. Though we consider partial MSCs with possibly unmatched events, a system description or an implementation of a system is often designed for complete message transfer, i.e., partial MSCs will be combined here towards basic MSCs.

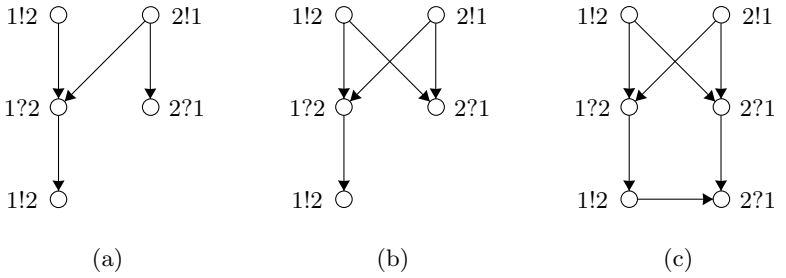


Fig. 7.2. Three sample partial MSCs

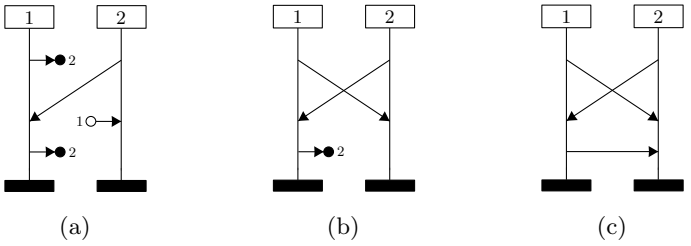


Fig. 7.3. The visual representation of partial MSCs

Observe that, for a partial MSC $\mathcal{M} = (V, \lhd, \lambda) \in \mathbb{PMSC}$ and an agent $i \in Ag$, the *projection* $\mathcal{M} \restriction i$ of $\mathcal{M}$ onto i is identical to $(V_i, \lhd_i, \lambda_{|V_i}) \in \mathbb{W}(\Gamma_i)$. Moreover, an MSC $\mathcal{M} \in \mathbb{MSC}$ is uniquely determined by the collection $(\mathcal{M} \restriction i)_{i \in Ag}$ of its projections, which does not apply to partial MSCs: $\mathbb{MSC}$ is projective whereas both $\mathbb{LMSC}$ and $\mathbb{PMSC}$ are not.

Given two partial MSCs $\mathcal{M} = (V, \lhd, \lambda)$ and $\mathcal{M}' = (V', \lhd', \lambda')$, let $\mathcal{M} \cdot \mathcal{M}' := (V'', \lhd'', \lambda'')$ be the *(asynchronous) concatenation* of $\mathcal{M}$ and $\mathcal{M}'$ where $V'' = V \cup V'$, $\lambda'' = \lambda \cup \lambda'$, and $\lhd''$ is the union of $\lhd$, $\lhd'$, and $\bigcup_{i \in Ag} \{(u, u') \in V_i \times V_i' \mid u = \mathit{last}(\mathcal{M} \restriction i) \text{ and } u' = \mathit{first}(\mathcal{M}' \restriction i)\}$. The attribute *asynchronous* is assigned because, after having concatenated two partial MSCs and considering their operational behavior in terms of the associated partial order, agents might decide independently of one another when to move from the first to the second factor. Observe that asynchronous concatenation is associative. As with M^--traces, we deal with $(\emptyset, \emptyset, \emptyset)$ as the unit MSC and denote it by $\mathbf{1}_{\mathbb{MSC}}$. We obtain the *monoids* $(\mathbb{PMSC}, \cdot, \mathbf{1}_{\mathbb{MSC}})$ and $(\mathbb{MSC}, \cdot, \mathbf{1}_{\mathbb{MSC}})$, which are identified with $\mathbb{PMSC}$ and $\mathbb{MSC}$, respectively. Some definitions (such as the one of communicating finite-state machines from Chap. 8) are for simplicity rather designed for nonempty MSCs. We will, however, include $\mathbf{1}_{\mathbb{MSC}}$ implicitly and it will always be clear how to cope with $\mathbf{1}_{\mathbb{MSC}}$ as well. Note that $(\mathbb{LMSC}, \cdot, \mathbf{1}_{\mathbb{MSC}})$ is a monoid, too, which, however, is not needed in the following.

Remark 7.6. Both $\mathbb{PMSC}$ and $\mathbb{MSC}$ are not finitely generated.

For a partial MSC $\mathcal{M} = (V, \lhd, \lambda) \in \mathbb{PMSC}$ over Ag, we define the *communication graph* of $\mathcal{M}$, denoted by $\mathrm{CG}(\mathcal{M})$, to be the pair $(Ag(\mathcal{M}), Arcs)$, $Arcs \subseteq Ag(\mathcal{M}) \times Ag(\mathcal{M})$, where, for any $i, j \in Ag(\mathcal{M})$, $(i, j) \in Arcs$ if there are $u, v \in V$ such that $\lambda(u) = i!j$ and $\lambda(v) = j?i$ [35]. Thus, $\mathrm{CG}(\mathcal{M})$ reflects the communication structure of $\mathcal{M}$. Remarkably, if $\mathcal{M}$ is a basic MSC, then $\mathcal{M}$ is connected iff $\mathrm{CG}(\mathcal{M})$ is connected, which does not hold for partial MSCs in general. Figure 7.4 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.

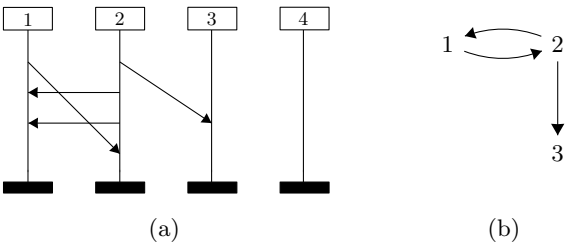


Fig. 7.4. A connected MSC and its communication graph

7.2 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. In general, we distinguish two kinds of boundedness. If we require any execution of an MSC (by which we mean a linearization) to correspond to a fixed channel capacity, we will speak of a *universally* bounded MSC [45]. If, in contrast, we require at least one linearization to fit into the channel restriction, we call an MSC *existentially* bounded [62]. 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 of whether an MSO formula satisfies a given *high-level MSC* (see Sect. 7.3) [35, 63, 64].

Let us become more formal and let $B \geq 1$. As we define boundedness in terms of linearizations of MSCs, we first call a word $\underline{w} \in \mathbb{W}(\Gamma)$ B-bounded if, for any prefix $\underline{u}$ of $\underline{w}$ and any $(i, j) \in Ch$, $|\underline{u}|_{i!j} - |\underline{u}|_{j?i} \leq B$. An MSC $\mathcal{M} \in \mathbb{MSC}$ is called

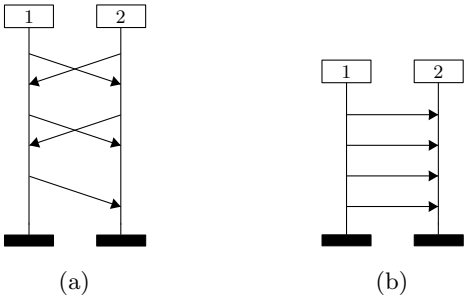


Fig. 7.5. A $\forall$2-bounded and an $\exists$1-bounded MSC

- *universally B-bounded* ($\forall B$-bounded) if, for any $\underline{w} \in Lin(\mathcal{M})$, $\underline{w}$ is B-bounded and
- *existentially B-bounded* ($\exists B$-bounded) if there is at least one $\underline{w} \in Lin(\mathcal{M})$ such that $\underline{w}$ 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 7.7. The MSC depicted in part (a) of Fig. 7.5 is $\forall$2-bounded because, at any time of execution, there are at most two messages in each channel: consider agent 1, which sends a message to agent 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 agent 1 is able to complete its agent line by sending a third message, it has to receive a second message from agent 2, which, in turn, is required to take at least one message from $(1, 2)$. As agent 1 can potentially send its second message without awaiting collection of the first one by agent 2, the MSC is not $\forall$1-bounded, though it is $\exists$1-bounded. The MSC depicted in Fig. 7.5b 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 number $B \geq 1$, the set of $\forall B$-/$\exists B$-bounded MSCs is denoted by $\mathbb{MSC}_{\forall B}/\mathbb{MSC}_{\exists B}$, respectively. An MSC language $L \subseteq \mathbb{MSC}$ will be called $\forall B$-/$\exists B$-bounded if, for any $\mathcal{M} \in L$, $\mathcal{M}$ 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.

7.3 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 $\mathcal{M}$ and $\mathcal{M}'$ is that, for any process line, events of $\mathcal{M}$ precede events of $\mathcal{M}'$, 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 [40]. 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 $\mathbb{PMSC}$ whose language, though consisting of partial MSCs, can be understood as a basic MSC language if previously unmatched events are combined according to a fifo architecture.

To make our life and upcoming constructions a bit easier, we replace in $\mathrm{REX}_{\mathbb{PMSC}}$ and $\mathrm{REX}_{\mathbb{MSC}}$ the standard operation α^* with α^+ ($L(\alpha^+)$ is still defined to be $L(\alpha)^+$) and consider α^* to serve as an abbreviation of $\alpha^+ + \mathbf{1}_{\mathbb{MSC}}$. Thus, a rational expression of $\mathbb{MSC}$ or $\mathbb{PMSC}$ comes up with the plus operation α^+ instead of α^*, which is not a restriction but circumvents some technical difficulties that come along with $\mathbf{1}_{\mathbb{MSC}}$.

Definition 7.8 (High-Level Compositional MSC, cf. [40]). *A* high-level compositional MSC *(HcMSC) (over Ag) is a rational expression of* $\mathbb{PMSC}(Ag)$.

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

In contrast to [40] and rather following [73], our approach to high-level (compositional) MSCs is algebraically motivated and, based on the idea of prime MSCs, allows us to relate the corresponding MSC languages to trace theory. Moreover, we follow the approach adopted in [3, 5, 45, 63, 74], where high-level MSCs are flattened into *message sequence graphs*.

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

An HcMSC might be represented as a (labeled) graph $(Q, \Delta, Q^{in}, F, \mu)$ where Q is the nonempty finite set of *states*, $\Delta \subseteq Q \times Q$ is the *step relation*, $Q^{in} \subseteq Q$ is the nonempty set of *initial states*, $F \subseteq Q$ is the set of *final states*, and μ is a mapping $Q \to \mathbb{PMSC}$. An *execution* of $\mathcal{H}$ is henceforth a sequence $q_0 \dots q_n \in Q^+$ of states such that $q_0 \in Q^{in}$ and $(q_i, q_{i+1}) \in \Delta$ for any $i \in \{0, \dots, n-1\}$. It is called *accepting* if, moreover, $q_n \in F$. An execution $\rho = q_0 \dots q_n$ of $\mathcal{H}$ gives rise to the partial MSC $\mathcal{M}(\rho) := \mu(q_0) \cdot \ldots \cdot \mu(q_n)$. Then, the language $L(\mathcal{H})$ can be defined to be $\{\mathcal{M}(\rho) \mid \rho \text{ is an accepting execution of } \mathcal{H}\}$. Observe that this view of an HcMSC is equivalent to the one proposed before. To see this, one needs to realize that a rational expression and a structure $(Q, \Delta, Q^{in}, F, \mu)$ both exhibit regular languages over finite alphabets, which arise from the partial MSCs that occur in the expression and in $\mu(Q)$, respectively. Given a rational expression $\mathcal{H}$ of $\mathbb{PMSC}$, the *graph* of $\mathcal{H}$, denoted by $\mathcal{G}raph(\mathcal{H})$, is inductively defined as follows (suppose α and β to be HcMSCs with graphs $(Q_\alpha, \Delta_\alpha, Q^{in}_\alpha, F_\alpha, \mu_\alpha)$ and $(Q_\beta, \Delta_\beta, Q^{in}_\beta, F_\beta, \mu_\beta)$):

- $\mathcal{G}raph(\emptyset) := (\{*\}, \emptyset, \{*\}, \emptyset, * \mapsto \mathbf{1}_{\mathbb{MSC}})$,
- $\mathcal{G}raph(\mathcal{M}) := (\{*\}, \emptyset, \{*\}, \{*\}, * \mapsto \mathcal{M})$,
- $\mathcal{G}raph(\alpha + \beta) := (Q_\alpha \cup Q_\beta, \Delta_\alpha \cup \Delta_\beta, Q^{in}_\alpha \cup Q^{in}_\beta, F_\alpha \cup F_\beta, \mu_\alpha \cup \mu_\beta)$,
- $\mathcal{G}raph(\alpha \cdot \beta) := (Q_\alpha \cup Q_\beta, \Delta_\alpha \cup \Delta_\beta \cup (F_\alpha \times Q^{in}_\beta), Q^{in}_\alpha, F_\beta, \mu_\alpha \cup \mu_\beta)$,
- $\mathcal{G}raph(\alpha^+) := (Q_\alpha, \Delta_\alpha \cup (F_\alpha \times Q^{in}_\alpha), Q^{in}_\alpha, F_\alpha, \mu_\alpha)$.

In fact, we have $L(\mathcal{H}) = L(\mathcal{G}raph(\mathcal{H}))$ for any HcMSC $\mathcal{H}$.

Definition 7.9 (Safe and Left-Closed HcMSC). *We call an HcMSC $\mathcal{H}$* safe *if, for any $\mathcal{M} \in L(\mathcal{H})$, there is some basic MSC $\mathcal{M}' \in \mathbb{MSC}$ such that $\mathcal{M} \preceq \mathcal{M}'$. We call it* left-closed *if, for any execution ρ of $\mathcal{G}raph(\mathcal{H})$, there is some lossy MSC $\mathcal{M}' \in \mathbb{LMSC}$ such that $\mathcal{M}(\rho) \preceq \mathcal{M}'$.*

Lemma 7.10 ([35]). *For any safe HcMSC $\mathcal{H}$, $\mathcal{L}(\mathcal{H})$ is $\exists$-bounded.*

Exercise 7.11. Show that any left-closed HcMSC is equivalent to some HcMSC that is both safe and left-closed.

Many interesting properties are undecidable for sets of MSCs formalized by 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 cooperativity* in high-level MSCs was introduced independently in [37] and [73] and extended to HcMSCs in [35].

Definition 7.12 (Globally Cooperative HcMSC). *We call an HcMSC $\mathcal{H}$* globally cooperative *(gc-HcMSC) if, for any subexpression β^+ of $\mathcal{H}$ and any $\mathcal{M} \in L(\beta)$, $\mathcal{M}$ is connected.*

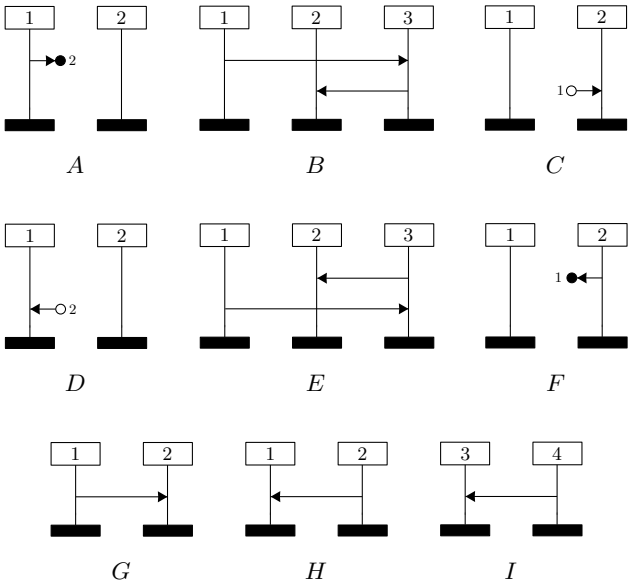


Fig. 7.6. The building blocks of HcMSCs

Thus, in a gc-HcMSC, iteration occurs over sets of connected partial MSCs only. Note that, actually, [35] 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 a 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 7.13 (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* $\mathbb{MSC}$.

An HMSC is trivially safe and left-closed. Moreover, the MSC language $\mathfrak{L}(\mathcal{H}) = L(\mathcal{H})$ of some HMSC $\mathcal{H}$ is finitely generated.

Example 7.14. Consider the partial MSCs from Fig. 7.6 and the following HcMSCs, whose graphs are illustrated in Fig. 7.7.

- The gc-HcMSC $\mathcal{H}_\mathrm{a} = A^+ \cdot B \cdot C^+$, whose basic MSC language features, among others, the basic MSC $\mathcal{M}_\mathrm{a}$ from Fig. 7.8, is neither safe nor left-closed.
- 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 $\mathcal{M}_\mathrm{b}$ from Fig. 7.8.

- $\mathcal{H}_{\mathrm{c}} = C \cdot E^{+} \cdot A$ is a safe, though not left-closed, gc-HcMSC, which defines the set of basic MSCs in the style of $\mathcal{M}_{\mathrm{c}}$ from Fig. 7.8. Note that there is no left-closed HcMSC equivalent to $\mathcal{H}_{\mathrm{c}}$.
- $\mathcal{H}_{\mathrm{d}} = A \cdot B^{+} \cdot C$ is a gc-HcMSC that is both safe and left-closed.
- 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.

Observe that $\mathcal{H}_{\mathrm{a}}$ and $\mathcal{H}_{\mathrm{b}}$ are both *not* equivalent to some safe HcMSC. In particular, HMSCs are in general not capable of defining the MSC language of the HcMSC $\mathcal{H}_{\mathrm{a}}$ from the above example, which defines MSCs in the style of $\mathcal{M}_{\mathrm{a}}$ from Fig. 7.8. 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 some language classes associated with HcMSCs and their restrictions. We set

$$\begin{aligned}
\mathcal{HcMSC} &:= \{\mathcal{L}(\mathcal{H}) \mid \mathcal{H} \text{ is an HcMSC}\} \\
\text{gc-}\mathcal{HcMSC} &:= \{\mathcal{L}(\mathcal{H}) \mid \mathcal{H} \text{ is a gc-HcMSC}\} \\
\text{safe-gc-}\mathcal{HcMSC} &:= \{\mathcal{L}(\mathcal{H}) \mid \mathcal{H} \text{ is a safe gc-HcMSC } \mathcal{H}\} \\
\text{left-closed-gc-}\mathcal{HcMSC} &:= \{\mathcal{L}(\mathcal{H}) \mid \mathcal{H} \text{ is a left-closed gc-HcMSC } \mathcal{H}\} \\
\mathcal{HMSC} &:= \{\mathcal{L}(\mathcal{H}) \mid \mathcal{H} \text{ is an HMSC}\} \\
\text{gc-}\mathcal{HMSC} &:= \{\mathcal{L}(\mathcal{H}) \mid \mathcal{H} \text{ is a gc-HMSC}\}
\end{aligned}$$

We obtain $\mathcal{HMSC} = \mathcal{RAT}_{\mathbb{MSC}}$, as any HMSC is a rational expression of $\mathbb{MSC}$ and vice versa.

Lemma 7.15. *The classes of HcMSCs form the hierarchy depicted in Fig. 7.9. The hierarchy is strict.*

Proof. Inclusions follow directly from the definitions. Moreover, strictness is witnessed by the HcMSCs from Example 7.14, which are assigned in Fig. 7.9 to a class they belong to so that there is no equivalent counterpart in a lower class, respectively. □

Note that none of the above classes will later ensure that a system is implementable without deadlocks. Consider, for example, the simple gc-HMSC $G + H$ (cf. Fig. 7.6) generating $\{G, H\}$ and suppose we are looking for a distributed implementation thereof. Unless we assume the implementation to have some global information at the beginning of a run so that agents 1 and 2 could agree on either G or H, both 1 and 2 may try to send a message to the other agent and thereupon switch to idle mode so that none of the messages is ever received.

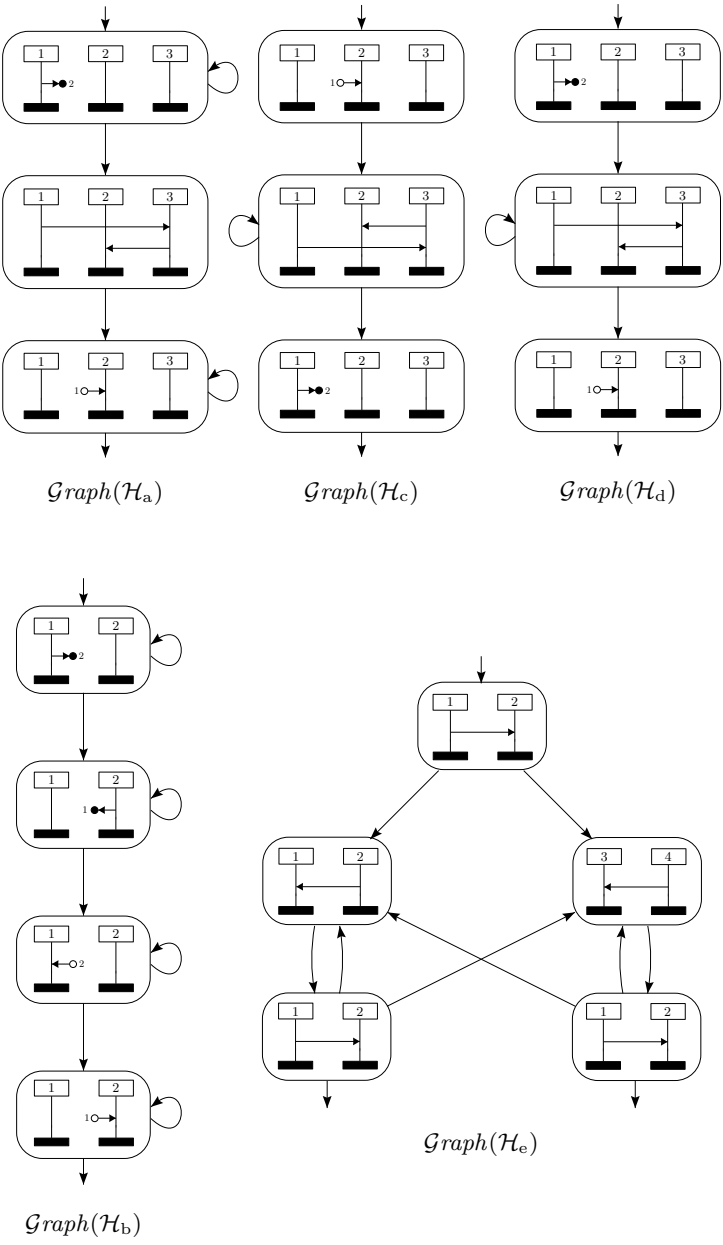


Fig. 7.7. HcMSCs as graphs

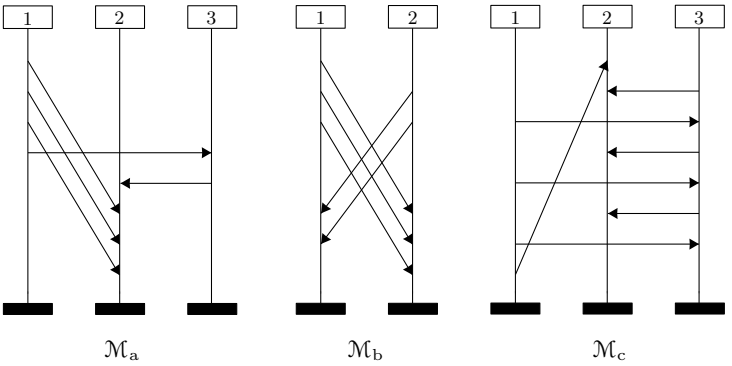


Fig. 7.8. The basic MSC languages of some gc-HcMSCs

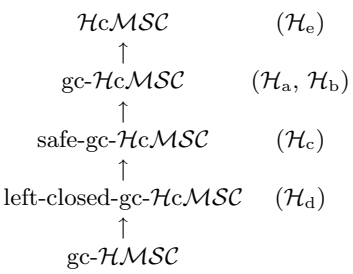


Fig. 7.9. The hierarchy of HcMSCs

Thus, the specification often admits a global view of the system, whereas a locally controlled implementation lacks some global information so that inconsistent local decisions might cause an undesired behavior (e.g., a deadlock). The class of *local-choice* HcMSCs rules out such misunderstanding and makes sure that, at any time of execution, there is a leader agent that decides on the further system behavior.

Definition 7.16 (Local-Choice HcMSC). *We call an HcMSC $\mathcal{H}$ with $\mathcal{G}raph(\mathcal{H}) = (Q, \Delta, Q^{in}, F, \mu)$* local-choice *if*

- *$\mathcal{H}$ is safe,*
- *there is an agent $i \in Ag$ such that, for any $q \in Q^{in}$ (set $(V, \lhd, \lambda)$ to be $\mu(q)$), there is a node $u \in V_i$ that is minimal in $(V, \leq)$, and*
- *for any $(q, q') \in \Delta$ (set $(V, \lhd, \lambda)$ to be $\mu(q')$), there exist an agent $i \in Ag(\mu(q))$ and an event $u \in V_i$ such that u is minimal in $(V, \leq)$.*

The corresponding class of MSC languages will be denoted by lc-$\mathcal{H}c\mathcal{MSC}$.

According to that definition, $G+H$ and $(G+H)^+$ are both not local-choice whereas $G \cdot (G+H)^+$ is (where, again, G and H are taken from Fig. 7.6). In the latter HMSC, agent 1 becomes the first leader in an implementation thereof and may send a message to inform agent 2 what to do next, i.e., if it should send or receive a message. In further steps, the sender respectively decides whether to realize G or H. To get an impression of what an automata implementation of the local-choice HMSC $G + G \cdot (G+H)^+$ looks like, the reader may consult Fig. 8.5 on page 123. Observe that the only local-choice HcMSC in Example 7.14 is $\mathcal{H}_{\mathrm{d}}$. Obviously, if we restrict to left-closed HcMSCs, local-choice HcMSCs are weaker than any other class proposed in this section. However, they provide a sufficient, though not necessary condition for implementability without deadlocks (cf. Chap. 8).

Note that algorithms for checking an HcMSC for the properties introduced in this section reduce to algorithms for processing its graph representation. Examining whether an HcMSC is safe can be done in linear time. Checking if an HcMSC is globally cooperative is co-NP complete [76], whereas it can be checked in polynomial time if a given HcMSC is local-choice (see [37] for an overview). Moreover, there are a couple of related complexity results. For example, Genest has recently shown that checking if some safe HcMSC is equivalent to some lc-HcMSC is in co-NP [34].

There is a noncommercial, freely available tool for checking MSC specifications for many of the above-mentioned properties [15]. It offers facilities to create MSC and HcMSC documents and to visualize and analyze them through a graphical user interface.

7.4 Message Contents and Non-Fifo Behavior

So far, we have not considered which message is actually sent when 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 with respect to Ag and a nonempty finite set Λ of *message contents*. An action from $\Gamma_i^s(Ag,\Lambda)$ is henceforth a symbol $i!^a j$, which indicates that a message $a \in \Lambda$ is sent from agent i to some agent j, and an action from $\Gamma_i^r(Ag,\Lambda)$ is a symbol $i?^a j$, which stands for receiving a so that we will write $(i!^a j, j?^a i) \in Com(Ag,\Lambda)$. As usual, the union of $\Gamma_i^s(Ag,\Lambda)$ and $\Gamma_i^r(Ag,\Lambda)$ is denoted by $\Gamma_i(Ag,\Lambda)$, that of $\Gamma^s(Ag,\Lambda)$ and $\Gamma^r(Ag,\Lambda)$ is denoted by $\Gamma(Ag,\Lambda)$. We canonically define the underlying distributed alphabet to be $\widetilde{\Gamma}(Ag,\Lambda) := (\Gamma_i(Ag,\Lambda))_{i\in Ag}$.

However, there is some scope for the canonical extension of the definition of an MSC and we might choose between modeling a fifo or non-fifo channel system. In both cases, we consider the following definition to be a starting point. Hereby, let us define basic MSCs directly without going via partial MSCs first.

Definition 7.17 (Extends Definition 7.3). *A* message sequence chart *over* (Ag, Λ) *is a structure* $\mathcal{M} = (V, \lhd, \lambda)$ *from* $\mathbb{DAG}(\widetilde{\Gamma}(Ag, \Lambda))$ *such that*

- *for any* $i \in Ag$, $\lhd_i = \lessdot_i$, *i.e.,* $\lhd_i$ *is the cover relation of* $\leq_i$,
- *for any* $u, v \in V$, $u \lhd_c v$ *iff* $(\lambda(u), \lambda(v)) \in Com(Ag, \Lambda)$ *and* $|\mathcal{M}\Downarrow u|_{\lambda(u)} = |\mathcal{M}\Downarrow v|_{\lambda(v)}$, *and*
- $|\mathcal{M}|_{i!^a j} = |\mathcal{M}|_{j?^a i}$ *for any* $(i, j) \in Ch$ *and* $a \in \Lambda$.

If we left Definition 7.17 as it currently stands, we would allow non-fifo-MSCs as depicted in Fig. 7.10 where messages of different type overtake each other in channel $(1, 2)$. In particular, an MSC over (Ag, Λ) is not necessarily a fifo-dag. We therefore denote the set of (potentially non-fifo) MSCs over (Ag, Λ) by $\mathbb{MSC}_{\not\searrow}(Ag, \Lambda)$. Such a communication model is considered in [10, 61, 73]. 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. Accordingly, Alur et al. call structures in which overtaking of identical messages occurs *degenerate* [4]. To relate MSCs and their linearizations even in that case, which is however explicitly provided by the MSC standard, one needs to equip any position of a word with additional information, say, a time stamp, to recover the order in which messages have been sent. If we wish an MSC $\mathcal{M} = (V, \lhd, \lambda)$ over (Ag, Λ) to behave in a fifo manner, we require in addition that $\mathcal{M} \in \mathbb{DAG}_{\rightrightarrows}(\widetilde{\Gamma}(Ag, \Lambda))$. Then, for any $i, j \in Ag$, $u_1, u_2 \in V_i$, and $v_1, v_2 \in V_j$ with both $u_1 \lhd_c v_1$ and $u_2 \lhd_c v_2$, we have $u_1 \leq_i u_2$ iff $v_1 \leq_j v_2$. To refer to the set of those (fifo) MSCs over (Ag, Λ), we just write $\mathbb{MSC}(Ag, \Lambda)$. Thus, $\mathbb{MSC}(Ag, \Lambda) = \mathbb{MSC}_{\not\searrow}(Ag, \Lambda) \cap \mathbb{DAG}_{\rightrightarrows}(\widetilde{\Gamma}(Ag, \Lambda))$. Moreover, we have $\mathbb{MSC}(Ag, \Lambda) = \mathbb{MSC}_{\not\searrow}(Ag, \Lambda)$ iff Λ is a singleton set.

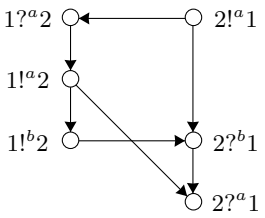


Fig. 7.10. An MSC with non-fifo behavior

However, if not otherwise explicitly stated, we will for simplicity deal with MSCs *over* Ag, abstracting away message contents. Exceptions are Sects. 7.5 and 8.5 as well as Theorem 8.34.

7.5 Live Sequence Charts

One might argue that HcMSCs are not yet flexible enough to be classified as an adequate specification language. For example, they cannot formalize forbidden behavior or distinguish between possible and necessary behavior. *Live sequence charts* (LSCs) largely correct those deficiencies [25]. They allow us to distinguish between *mandatory*, *provisional*, and *prohibited* behavior, which *must*, *may*, and *must not* happen, respectively. Hereby, mandatory progress is suggested by liveness conditions such as "whenever A occurs, the system has to act subsequently like B". Consider the left-hand side of Fig. 7.11, which depicts some *universal* LSC describing the part of a system where a coin is flipped several times in a row until the outcome is "heads". Referring to the above liveness requirement, behavior A (flipping the coin) is enclosed within a dashed borderline and called the *prechart*, whereas B (the coin is either "heads" or "tails"), which is enforced to happen whenever A occurs, is enclosed within a solid line and called the *chart body*.

For illustrative purposes, we consider MSCs over (Ag, Λ) in this section, where Λ is a given message alphabet. Let us, however, simply write $\mathbb{MSC}$ instead of $\mathbb{MSC}(Ag, \Lambda)$.

Definition 7.18 (Live Sequence Chart).

- *A* universal LSC *(over (Ag, Λ)) is a pair* $(\mathcal{M}, L) \in \mathbb{MSC} \times 2^{\mathbb{MSC}}$ *(often written as* $\Box(\mathcal{M}, L)$ *in this context).*
- *An* existential LSC *(over (Ag, Λ)) is an MSC* $\mathcal{M} \in \mathbb{MSC}$ *(often written as* $\Diamond(\mathcal{M})$ *in this context).*

Thus, an LSC can be of two different types: it can be either universal or existential. This nomenclature leads one to assume its meaning for the system behavior. In fact, a universal LSC $\Box(\mathcal{M}, L)$ is required to be satisfied by *all* system runs, whereas the behavior imposed by an existential LSC must occur in *at least one* possible run of the system.

An *LSC specification* (over (Ag, Λ)) is a pair $\mathcal{S} = (\forall\mathcal{S}, \exists\mathcal{S})$ with $\forall\mathcal{S}$ a finite set of universal LSCs and $\exists\mathcal{S}$ a finite set of existential LSCs. Unlike HcMSCs, $\mathcal{S}$ does not determine *the one* system in terms of its MSC language. We rather say that an MSC language either satisfies $\mathcal{S}$ or not. Accordingly, $\mathcal{S}$ might admit several different languages. Formally, we say that an MSC language $L_{sys} \subseteq \mathbb{MSC}$ *satisfies* $\mathcal{S}$, written $L_{sys} \models \mathcal{S}$, if both

- for any $\Box(\mathcal{M}, L) \in \forall\mathcal{S}$ and any $\mathcal{M}_1, \mathcal{M}_2 \in \mathbb{MSC}$, $\mathcal{M}_1 \cdot \mathcal{M} \cdot \mathcal{M}_2 \in L_{sys}$ implies $\mathcal{M}_2 \in L \cdot \mathbb{MSC}$ and
- for any $\Diamond(\mathcal{M}) \in \exists\mathcal{S}$, there are $\mathcal{M}_1, \mathcal{M}_2 \in \mathbb{MSC}$ such that $\mathcal{M}_1 \cdot \mathcal{M} \cdot \mathcal{M}_2 \in L_{sys}$.

Of course, an LSC specification $\mathcal{S}$ is desired to determine at least one possible system so that we call $\mathcal{S}$ consistent if $\{L_{sys} \subseteq \mathbb{MSC} \mid L_{sys} \models \mathcal{S}\}$ is not the empty set.

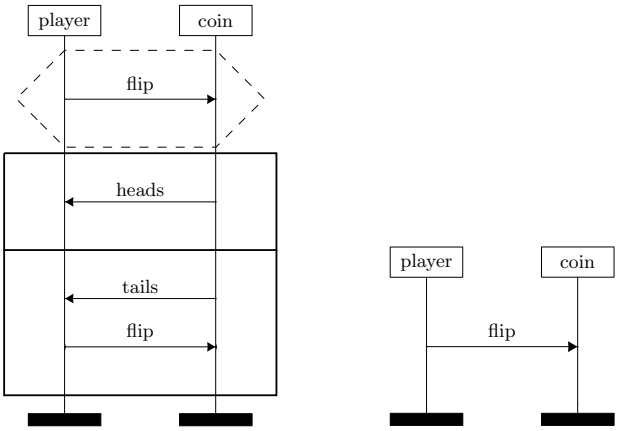


Fig. 7.11. An LSC specification

Example 7.19. We suppose that $\mathcal{S}$ is the LSC specification over the pair ({player,coin}, {flip,heads,tails}) that is depicted in Fig. 7.11, which consists of one universal LSC (whose second component contains two elements) and one existential one. Then, $\mathcal{S}$ is consistent, as it is, for example, satisfied by any nonempty subset of the MSC language that is illustrated in Fig. 7.12. In particular, it is satisfied by the singleton set containing the MSC in which a coin is flipped once and shows "heads".

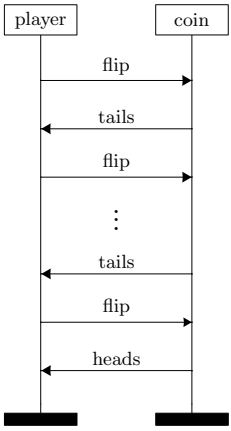


Fig. 7.12. A coin-flip scenario

7.6 Regular MSC Languages

There have been several proposals for the *right* notion of regularity for MSC languages. In their seminal work [45], 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 Fig. 7.6 on page 99), which allows us to concatenate G arbitrarily often, is not $\forall$-bounded and hence cannot be regular. It induces the set of MSCs that send arbitrarily many messages from agent 1 to agent 2. The corresponding set of linearizations gives rise to a set of words that, in particular, exhibit 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. Otherwise, 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 7.20 (Regular MSC Language [45]). *An MSC language $L \subseteq \mathbb{MSC}$ is called* regular *if $Lin(L) \in \mathcal{REC}_{\mathbb{W}(\Gamma)}$, i.e., if $Lin(L)$ is a regular word language over Γ.*

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

Lemma 7.21 ([45]). *Any regular MSC language is $\forall$-bounded.*

Exercise 7.22. Prove Lemma 7.21.

As mentioned above, $\{G\}^*$ with G again taken from Fig. 7.6 is not regular. However, it is $\exists 1$-bounded and there is a simple gc-HMSC defining this language. Moreover, there is an asynchronous cellular automaton (and, in terms of MSCs, a communicating finite-state machine) accepting $\{G\}^*$. Thus, there are EMSO definable languages that are not regular. We therefore consider EMSO definability to be another criterion for *regularity*. So let us in the next section examine (existential) MSO logic whose formulas are interpreted over MSCs.

7.7 (E)MSO-Definable MSC Languages

Recall that an MSC is modeled as a graph, which corresponds to the view taken in [17, 63] and also adopted by [12, 57] where (the graph of) an MSC is called a *message flow graph*. However, while most theorems hold independently of the modeling, the way to define an MSC immediately affects the syntax and expressivity of (fragments of) the corresponding MSO logic. As $\mathbb{MSC}(Ag) \subseteq \mathbb{DAG}(\Gamma(Ag))$, the MSO formulas that can be applied to MSCs are those from $\mathrm{MSO}(\Gamma(Ag))$. Recall that the corresponding atomic entities are

$$\lambda(x) = \sigma \qquad x \lhd y \qquad x \in X \qquad x = y$$

(where $\sigma \in \Gamma$, $x, y \in \mathrm{Var}$ and $X \in \mathrm{VAR}$). The definition of their semantics arises from the general case of graphs. Recall that the class of $\mathrm{MSO}_{\mathbb{MSC}}$-definable MSC languages is denoted by $\mathcal{MSO}_{\mathbb{MSC}}$, that of $\mathrm{EMSO}_{\mathbb{MSC}}$-definable MSC languages by $\mathcal{EMSO}_{\mathbb{MSC}}$ and so on. Given $i \in Ag$ and an individual variable x, the formula $Ag(x) = i$ will throughout the book stand for $\lambda(x) \in \Gamma_i$ (which, in turn, was a shorthand for $\bigvee_{\sigma \in \Gamma_i} \lambda(x) = \sigma$).

Example 7.23. Let L be the set of MSCs over $\{1, 2, 3\}$ whose linearizations are of the form $(1!2)^n(1!3)(3?1)(3!2)(2?3)(2?1)^n$, $n \in \mathbb{N}$. Note that $L \in \text{gc-}\mathcal{HcMSC}$ and that an example MSC from L is provided by $\mathcal{M}_{\mathrm{a}}$ from Fig. 7.8 on page 102. 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 agents 1, 2, and 3, respectively, and are given as follows:

$$\begin{aligned}
\varphi_1 = \exists x(\quad & \lambda(x) = 1!3 \\
& \wedge \forall y(Ag(y) = 1 \wedge \neg(x = y) \rightarrow \lambda(y) = 1!2) \\
& \wedge \neg\exists y(x \lhd_1 y))
\end{aligned}$$

$$\begin{aligned}
\varphi_2 = \exists x(\quad & \lambda(x) = 2?3 \\
& \wedge \forall y(Ag(y) = 2 \wedge \neg(x = y) \rightarrow \lambda(y) = 2?1) \\
& \wedge \neg\exists y(y \lhd_2 x))
\end{aligned}$$

$$\begin{aligned}
\varphi_3 = \exists x(\quad & \lambda(x) = 3?1 \\
& \wedge \neg\exists y(y \lhd_3 x))
\end{aligned}$$

So if we let $\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3$, then $L_{\mathbb{MSC}}(\varphi) = L$. Observe that $\varphi \in \mathrm{FO}$ and therefore $L \in \mathcal{FO}_{\mathbb{MSC}}$. As L generates a set of total orders that gives rise to a non-regular word language over Γ, $Lin(L)$ is not $\mathrm{MSO}(\Gamma)_{\mathbb{W}(\Gamma)}$-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.

Exercise 7.24. Show that ($\mathcal{L}(\mathcal{H}_{\mathrm{b}})$–$\mathcal{L}(\mathcal{H}_{\mathrm{e}})$ are taken from Fig. 7.7):

(a) $\{\mathcal{M} \in \mathbb{MSC} \mid \mathcal{M} \text{ is connected}\} \in \mathcal{FO}_{\mathbb{MSC}}$,
(b) $\mathcal{L}(\mathcal{H}_{\mathrm{b}}) \in \mathcal{FO}_{\mathbb{MSC}(\{1,2\})}$,
(c) $\mathcal{L}(\mathcal{H}_{\mathrm{c}}) \in \mathcal{FO}_{\mathbb{MSC}(\{1,2,3\})}$,
(d) $\mathcal{L}(\mathcal{H}_{\mathrm{d}}) \in \mathcal{FO}_{\mathbb{MSC}(\{1,2,3\})}$,
(e) $\mathcal{L}(\mathcal{H}_{\mathrm{e}}) \notin \mathcal{EMSO}_{\mathbb{MSC}(\{1,2,3,4\})}$,
(f) $\mathcal{EMSO}[\{\leq_i\}_{i \in Ag}, \lhd_{\mathrm{c}}]_{\mathbb{MSC}} \subseteq \mathcal{EMSO}[\leq, \lhd_{\mathrm{c}}]_{\mathbb{MSC}}$,
(g) $\{\mathcal{M} \in \mathbb{MSC} \mid |\mathcal{M}|_{\Gamma^s} \text{ is even}\} \notin \mathcal{FO}_{\mathbb{MSC}}$,
(h) $\{\mathcal{M} \in \mathbb{MSC} \mid |\mathcal{M}|_{\Gamma^s} \text{ is even}\} \in \mathcal{EMSO}_{\mathbb{MSC}}$,

(i) the set of MSCs $(V, \lhd, \lambda) \in \mathbb{MSC}(\{1,2,3\})$ such that, for any $u, v \in V$ with $\lambda(u) = 1!2$ and $\lambda(v) = 1!3$, we have $u < v$ is not $\mathrm{FO}_{\mathbb{MSC}(\{1,2,3\})}$- but $\mathrm{EMSO}_{\mathbb{MSC}(\{1,2,3\})}$-definable.

There is an MSC language that is $\mathrm{MSO}_{\mathbb{MSC}}$-definable (even $\mathrm{MSO}[\leq]_{\mathbb{MSC}}$-definable) but not $\mathrm{EMSO}_{\mathbb{MSC}}$-definable. The proof of this separation result is addressed in Chap. 9.

7.8 Product MSC Languages

Languages defined by finite transition systems working in parallel are known as *product languages* and were initially studied by Thiagarajan [88] in the domain of Mazurkiewicz traces where distributed components communicate executing actions simultaneously rather than sending messages (cf. Chap. 6). Taking up the idea of product behavior, [4] considers MSC languages that are closed under *inference*, which can be described by the setting depicted in Fig. 7.6 on page 99. Attempting to realize the MSC language $\{G, I\}$, one might argue that the behavior of $G \cdot I$ is a feasible one, too. As agents 1 and 2 do not get in touch with processes 3 and 4, it is not clear to a single agent 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 [4], no finiteness condition was studied so that, in principle, it is possible to *realize* $\{G^{n^2} \mid n \in \mathbb{N}\}$. Summarizing, we may say that product behavior respects *independence*.

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

$$\forall i \in Ag : \ \exists \mathcal{M}' \in L : \ \mathcal{M}' \restriction i = \mathcal{M} \restriction i$$

Definition 7.25 (Weak Product MSC Language, cf. [4]). *A set $L \subseteq \mathbb{MSC}$ is called a* weak product MSC language *(over Ag) if, for any $\mathcal{M} \in \mathbb{MSC}$, $L \vdash_{Ag} \mathcal{M}$ implies $\mathcal{M} \in L$. The finite union of weak product MSC languages is called a* product MSC language.

Adopting the notation we introduced in the analogous framework 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 Fig. 7.6, $G \cdot I$ 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, whereas $\{G \cdot I\}^*$ is not.

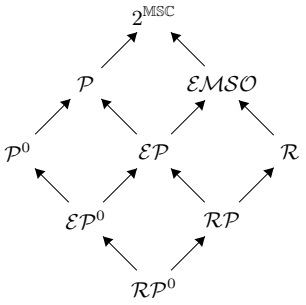


Fig. 7.13. The hierarchy of product MSC languages

Bringing together the concepts of product behavior and regularity, we call $\mathcal{RP}^0_{\mathbb{MSC}} := \mathcal{R}_{\mathbb{MSC}} \cap \mathcal{P}^0_{\mathbb{MSC}}$ the class of *weak regular product MSC languages* and $\mathcal{RP}_{\mathbb{MSC}}$ the class of *regular product MSC languages*, which is the closure of $\mathcal{RP}^0_{\mathbb{MSC}}$ under finite union.

Let us now extend our study towards product MSC 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 communicating finite-state machine, we rather concentrate on EMSO-definable languages than on MSO-definable ones. We call $\mathcal{EP}^0_{\mathbb{MSC}} := \mathcal{EMSO}_{\mathbb{MSC}} \cap \mathcal{P}^0_{\mathbb{MSC}}$ the class of *weak EMSO-definable product MSC languages*, and we denote by $\mathcal{EP}_{\mathbb{MSC}}$ the class of *EMSO-definable product MSC languages*, which is the closure of $\mathcal{EP}^0_{\mathbb{MSC}}$ under finite union.

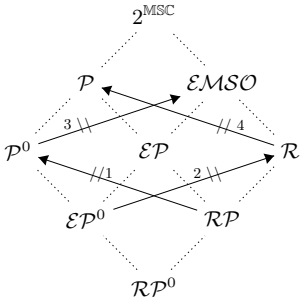
If it is clear from the context that we talk about MSCs, we omit the reference to $\mathbb{MSC}$ and simply write $\mathcal{R}$, $\mathcal{P}$, and $\mathcal{MSO}$. Then, we may also speak of regular or product languages.

Theorem 7.26. *The classes of languages proposed so far (apart from the class $\mathcal{HcMSC}$, which is reconsidered in Chap. 9 in more detail) draw the picture shown in Fig. 7.13. The hierarchy is strict.*

Proof. We will prove $\mathcal{R} \subseteq \mathcal{EMSO}$ in Chap. 8. The other inclusions are straightforward. It remains to show strictness and incomparability. Consider the MSCs G and I from Fig. 7.6 on page 99. For a (crossed) arrow from a class of MSC languages $\mathfrak{C}_1$ to a class $\mathfrak{C}_2$ in Fig. 7.14, the right-hand table specifies an MSC language L with $L \in \mathfrak{C}_1$ and $L \notin \mathfrak{C}_2$. □

7.9 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 and applies some relabeling



1	$\{G, I\}$
2	$\{G\}^*$
3	$L(f_k)$ with $k \geq 2$ (cf. Theorem 9.5)
4	$L_{\widehat{N}+1}$ with $\widehat{N} \geq 1$ (cf. Lemma 8.45)

Fig. 7.14. Strictness and incomparability in the hierarchy

to the events of an MSC to obtain a trace [56], whereas the other, basically considering several events of an MSC to be an event of a trace, is likewise applicable to unbounded behaviors [73].

In the following, let $\alpha \in \{+, -\}$ and let B be a positive natural number. We define a dependence alphabet $(\Gamma \times \{1, \ldots, B\}, D_B)$ where $(\sigma_1, n_1) D_B (\sigma_2, n_2)$ if we have

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

Note that, though the definitions from this section each depend on Ag, we omit a corresponding reference. Setting Co to be $\{(\sigma, \tau, n) \mid (\sigma, \tau) \in Com,\ n \in \{1, \ldots, B\}\}$, let $\widetilde{\Gamma}_B$ be the distributed alphabet $(\overline{\Gamma_\gamma})_{\gamma \in Ag \cup Co}$ where, for $i \in Ag$, $\overline{\Gamma}_i := \Gamma_i \times \{1, \ldots, B\}$ and, for $(\sigma, \tau, n) \in Co$, $\overline{\Gamma}_{(\sigma,\tau,n)} := \{(\sigma, n), (\tau, n)\}$. For example, if $Ag = \{1, 2\}$ and $B = 2$,

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

Remark 7.27. Given $B \geq 1$, we have $D_{\widetilde{\Gamma}_B} = D_B$.

To an MSC $\mathcal{M} = (V, \lhd, \lambda) \in \mathbb{MSC}_{\forall B}$, we assign the $\Gamma \times \{1, \ldots, B\}$-labeled poset $\mathcal{P}_B(\mathcal{M}) := (V', \leq', \lambda')$ where $V' = V$, $\leq' = \leq$, and, for each $u \in V$, we define $\lambda'(u)$ to be the new labeling $(\lambda(u), (|\mathcal{M}{\downarrow}u|_{\lambda(u)} \mod B) + 1)$.

Lemma 7.28 (Kuske [56]). *Let $B \geq 1$. For any MSC $\mathcal{M} \in \mathbb{MSC}_{\forall B}$, $\mathcal{P}_B(\mathcal{M})$ is a poset-trace over $\widetilde{\Gamma}_B$.*

Exercise 7.29. Prove Lemma 7.28.

Given $\mathcal{M} \in \mathbb{MSC}_{\forall B}$, we moreover denote by $Tr_B^\alpha(\mathcal{M})$ the M^α-trace of the poset-trace $\mathcal{P}_B(\mathcal{M})$. Note that the mapping $Tr_B^\alpha : \mathbb{MSC}_{\forall B} \to \mathbb{TR}^\alpha(\widetilde{\Gamma}_B)$ 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 $Tr_B^\alpha(L) \subseteq \mathbb{TR}^\alpha(\widetilde{\Gamma}_B)$.

Example 7.30. Taking $\mathcal{M} \in \mathbb{MSC}_{\forall 2}$ to be the MSC from Fig. 7.5a, the M^--trace $Tr_2^-(M)$ over $\widetilde{\Gamma}_2$ is shown in Fig. 7.15. Note that, in fact, the labelings of the events u and v, which are incomparable with respect to the partial order $\leq$ associated with $Tr_2^-(M)$, are independent as they do not occur together in some local alphabet of $\widetilde{\Gamma}_2$.

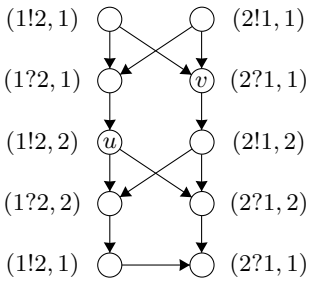


Fig. 7.15. The M^--trace of a 2-bounded MSC

In [35], the above relabeling is applied to existentially-bounded MSCs as well, which also gives rise to Mazurkiewicz traces if we add some edges between events that are actually independent in the MSC.

In the following, let us 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 [88, 89].

Proposition 7.31 ([45, 56]). *For any $\alpha \in \{+, -\}$, $B \geq 1$, and $L \subseteq \mathbb{MSC}_{\forall B}$,*

$$L \in \mathcal{R}_{\mathbb{MSC}} \quad \textit{iff} \quad Tr_B^\alpha(L) \in \mathcal{R}_{\mathbb{TR}^\alpha(\widetilde{\Gamma}_B)}$$

Proof. "Only if": Let $B \geq 1$ and let $L \subseteq \mathbb{MSC}_{\forall B}$ be a $\forall B$-bounded regular MSC language, i.e., $Lin(L)$ is recognized by some minimal finite automaton $\mathcal{A}$ over Γ (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 $\Gamma \times \{1,\ldots,B\}$ recognizing $Lin(Tr_B^\alpha(L))$.

"If": As shown in [45, 56], any asynchronous automaton over $\widetilde{\Gamma}_B$ has an equivalent counterpart in the form of a (strongly) $\forall$-bounded communicating finite-state machine (cf. Chap. 8). Together with Theorems 6.22 and 8.22, this proves the lemma. □

Proposition 7.32. *For any $\alpha \in \{+,-\}$, $B \geq 1$, and $L \subseteq \mathbb{MSC}_{\forall B}$,*

$$L \in \mathcal{RP}^0_{\mathbb{MSC}} \quad \textit{iff} \quad Tr_B^\alpha(L) \in \mathcal{RP}^0_{\mathbb{TR}^\alpha(\widetilde{\Gamma}_B)}$$

Proof. According to Proposition 7.31, the operator Tr_B^α 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 $Tr_B^\alpha(L)$ is a regular trace language over $\widetilde{\Gamma}_B = (\overline{\Gamma}_\gamma)_{\gamma \in Ag \cup Co}$. Moreover, let $\mathcal{T} \in \mathbb{TR}^\alpha(\widetilde{\Gamma}_B)$ such that, for any $\gamma \in Ag \cup Co$, there is a trace $\mathcal{T}_\gamma \in Tr_B^\alpha(L)$ satisfying $\mathcal{T}_\gamma \restriction \gamma = \mathcal{T} \restriction \gamma$. Then, $\mathcal{T} \in Tr_B^\alpha(\mathbb{MSC}_{\forall B})$ and, in particular, we have $\mathcal{T}_i \restriction i = \mathcal{T} \restriction i$ and, thus, $(Tr_B^\alpha)^{-1}(\mathcal{T}_i) \restriction i = (Tr_B^\alpha)^{-1}(\mathcal{T}) \restriction i$ for any $i \in Ag$, which implies $(Tr_B^\alpha)^{-1}(\mathcal{T}) \in L$ and $\mathcal{T} \in Tr_B^\alpha(L)$.

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

The proof of the following extension towards finite unions of weak regular product languages is left to the reader as an easy exercise.

Corollary 7.33. *For any $\alpha \in \{+,-\}$, $B \geq 1$, and $L \subseteq \mathbb{MSC}_{\forall B}$,*

$$L \in \mathcal{RP}_{\mathbb{MSC}} \quad \textit{iff} \quad Tr_B^\alpha(L) \in \mathcal{RP}_{\mathbb{TR}^\alpha(\widetilde{\Gamma}_B)}$$

A nonempty partial MSC $\mathcal{M}$ is called *prime* if $\mathcal{M} = \mathcal{M}_1 \cdot \mathcal{M}_2$ implies $\mathcal{M}_1 = \mathbf{1}_{\mathbb{MSC}}$ or $\mathcal{M}_2 = \mathbf{1}_{\mathbb{MSC}}$. Consider Fig. 7.16, for example. The partial MSCs from parts (a) and (d) are prime, while the partial MSCs in between are not. For the rest of this section, let Π 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 [43], gives rise to a natural dependence relation based on the distributed alphabet $\widetilde{\Sigma}_\Pi := (\Sigma_i)_{i \in Ag}$ where, for any $i \in Ag$, $\Sigma_i = \{\mathcal{M} \in \Pi \mid i \in Ag(\mathcal{M})\}$. We may accordingly declare prime partial MSCs $\mathcal{M}$ and $\mathcal{M}'$ independent if $Ag(\mathcal{M}) \cap Ag(\mathcal{M}') = \emptyset$.

Lemma 7.34. *For $\alpha \in \{+,-\}$, the morphism $\mathfrak{R}_\Pi : \mathbb{TR}^\alpha(\widetilde{\Sigma}_\Pi) \to \langle \Pi \rangle_{\mathbb{PMSC}}$, which maps an M^α-trace over $\widetilde{\Sigma}_\Pi$ with linearization $a_1 \ldots a_n$ onto the partial MSC $a_1 \cdot \ldots \cdot a_n$, is an isomorphism.*

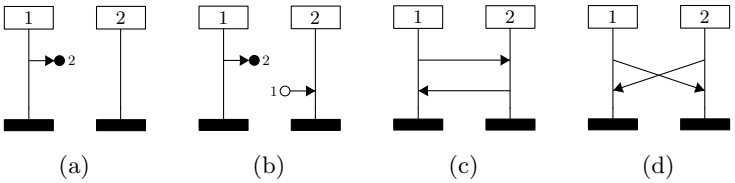


Fig. 7.16. Some prime and some non-prime partial MSCs

The proof is similar to a proof from [73] where the corresponding result is shown for basic MSCs rather than partial MSCs.

A basic MSC may correspond to several traces, i.e., there might be distinct traces $\mathcal{T}_1, \mathcal{T}_2 \in \mathbb{TR}^{\alpha}(\widetilde{\Sigma}_{\Pi})$ such that $\mathfrak{R}_{\Pi}(\mathcal{T}_1)$ and $\mathfrak{R}_{\Pi}(\mathcal{T}_2)$ represent the same MSC. This applies, for example, to the M^{+}-traces from Fig. 7.17: though they are different, they represent one and the same basic MSC. However, in our main application of prime partial MSCs in Chap. 9, those traces are not distinguished anyway.

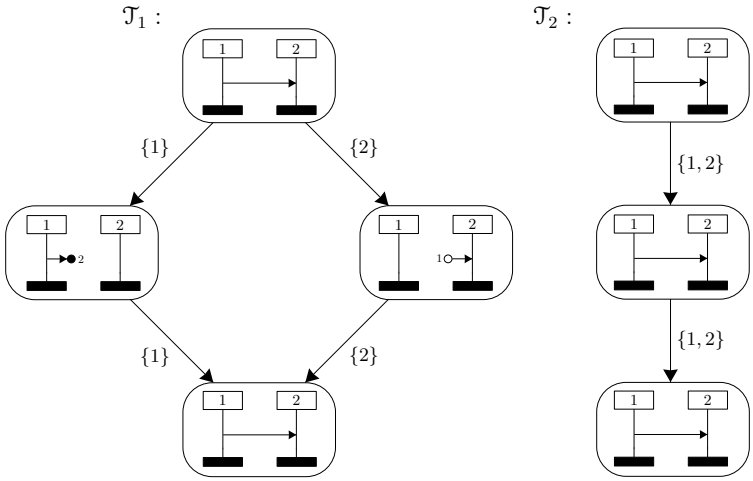


Fig. 7.17. Two trace representations of one basic MSC

7.10 Bibliographic Notes

One of the first papers aiming at giving MSCs a formal semantics was [67], which described the behavior of an MSC in a process algebra. To make MSCs

accessible to automata theory and logics, however, it pays to consider an MSC to be a partial order [52] or a graph [12], whose entities are handled by automata more naturally. Summarizing, we model an MSC as a graph, adopting the view taken in [12, 17, 57, 63] rather than considering partial orders [45, 52, 56, 73].

While we first proposed HcMSCs as the most general case of our specification language, HMSCs were actually the starting point of everything [68]. The study of compositionality was then initiated in [40] by Gunter et al. Note that, in [40], left-closed HcMSCs are called *realizable*. The notion of global cooperativity was independently introduced by Morin in [73] and Genest et al. in [37], who also proposed the notion of *local cooperativity* and showed that any local-choice HMSC is implementable without deadlocks. Since then, several extensions have been considered to facilitate and enrich the specification of communicating systems. For instance, [59] introduces a formalism that allows for generating processes during system execution, while we assume the number of processes to be fixed. In [11] and [75], *netcharts* are proposed and studied, which combine HMSCs with Petri nets to generate executable behavior. A formalism that is related to LSCs but follows a different approach is that of *triggered message sequence charts* [87], which provide means of expressing conditional and partial behavior. Another important area, which, however, is not considered in this book, is verification of specifications against formalisms such as temporal logics [36, 69, 70], MSO logics [59, 63, 64], and *template* MSCs [36, 78]. For a general introduction to that field, in particular model checking, the reader may be referred to [23].

8

Communicating Finite-State Machines

In this chapter, we introduce and study *communicating machines* (CMs), a model of computation rather than a specification language, which is close to a real-life implementation of a communicating system where distributed components communicate via fifo channels (which might be reliable or faulty, bounded or unbounded).

8.1 Communicating (Finite-State) Machines

A CM is a collection of state machines that share one global initial state and several global final states. The machines are connected pairwise with (for the moment) unbounded reliable fifo buffers. The transitions of each component are labeled with send or receive actions. Hereby, a send action $i!j$ puts a message at the end of the channel from i to j. A receive action can be taken provided the requested message is found in the channel. To extend the expressive power, CMs can send certain *synchronization messages*. Let us be more precise:

Definition 8.1 (Communicating Machine). *A* communicating machine *(over Ag) is a structure*

$$\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \mathcal{D}, \overline{s}^{in}, F)$$

such that

- $\mathcal{D}$ *is a nonempty finite set of* synchronization messages *(or* data*),*
- *for each $i \in Ag$, $\mathcal{A}_i$ is a pair (S_i, Δ_i) where*
  - S_i *is a nonempty set of (i-)*local states *and*
  - $\Delta_i \subseteq S_i \times \Gamma_i(Ag) \times \mathcal{D} \times S_i$ *is the set of (i-)*local transitions*,*
- $\overline{s}^{in} \in \prod_{i \in Ag} S_i$ *is the* global initial state*, and*
- $F \subseteq \prod_{i \in Ag} S_i$ *is the finite set of* global final states*.*

A CM $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, is called

- an N-CM, $N \in \mathbb{N}_{\geq 1}$, if $|\mathcal{D}| = N$,
- a *communicating finite-state machine* (CFM) or *finite* if, for each $i \in Ag$, S_i is finite,
- *locally accepting* if, for any $i \in Ag$, there is a set $F_i \subseteq S_i$ such that $F = \prod_{i\in Ag} F_i$, and
- *deterministic* if, for any $i \in Ag$, Δ_i satisfies the following conditions:
  - If $(s, i!j, m_1, s_1) \in \Delta_i$ and $(s, i!j, m_2, s_2) \in \Delta_i$, then $m_1 = m_2$ and $s_1 = s_2$.
  - If $(s, i?j, m, s_1) \in \Delta_i$ and $(s, i?j, m, s_2) \in \Delta_i$, then $s_1 = s_2$.

By CM(Ag), we denote the class of CMs over Ag and by CFM(Ag) the class of CFMs.[1] However, as the underlying set of agents will be clear from the context, we henceforth omit any reference to Ag and just write CM and, respectively, CFM. For a set $\mathfrak{C}$ of CMs, we denote by N-$\mathfrak{C}$, $\mathfrak{C}_\ell$, and det-$\mathfrak{C}$ the classes of N-, locally accepting, and deterministic CMs $\mathcal{A}$ with $\mathcal{A} \in \mathfrak{C}$, respectively.

We now define the behavior of CMs and, in doing so, adhere to the style of [56]. In particular, an automaton will run on MSCs rather than linearizations of MSCs, allowing for its distributed behavior. Let $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be a CM and moreover let $\mathcal{M} = (V, \lhd, \lambda) \in \mathbb{MSC}$ be an MSC. A *run* of $\mathcal{A}$ on $\mathcal{M}$ is a pair (ρ, μ) of mappings $\rho : V \to \bigcup_{i\in Ag} S_i$ with $\rho(u) \in S_{Ag(u)}$ for each $u \in V$ and $\mu : V \to \mathcal{D}$ such that

- for any $u, v \in V$ with $u \lhd_{\mathrm{c}} v$, $\mu(u) = \mu(v)$, and
- for any $u \in V$, $(source^{\overline{s}^{in}}_{(\mathcal{M},\rho)}(u)[Ag(u)], \lambda(u), \mu(u), \rho(u)) \in \Delta_{Ag(u)}$.

We call (ρ, μ) *accepting* if $final^{\overline{s}^{in}}_{(\mathcal{M},\rho)} \in F$. By $L(\mathcal{A}) := \{\mathcal{M} \in \mathbb{MSC} \mid$ there is an accepting run of $\mathcal{A}$ on $\mathcal{M}\}$, let us denote the *language* of $\mathcal{A}$. In particular, $\mathbf{1}_{\mathbb{MSC}}$ is a member of $L(\mathcal{A})$ if (and only if) $\overline{s}^{in} \in F$. Given a class $\mathfrak{C}$ of CMs, we furthermore set $\mathcal{L}(\mathfrak{C})$ to be $\{L \subseteq \mathbb{MSC} \mid$ there is $\mathcal{A} \in \mathfrak{C}$ such that $L(\mathcal{A}) = L\}$, which is the class of languages of $\mathfrak{C}$. We consider $\mathcal{L}$(CFM) to be some kind of standard class, which is identified by $\mathcal{CFM} := \mathcal{L}$(CFM). We also say that the languages from $\mathcal{CFM}$ are the *implementable* ones. This nomenclature is arbitrary and rather geared to the literature, where the term *realizability* usually refers to locally accepting 1-CMs. The intuition behind local acceptance is that recognition by the whole system defers to acceptance by any single component. In other words, any component may decide independently of the others whether it accepts or not. In contrast, the general acceptance condition allows to take a global view of the system to decide upon acceptance.

Example 8.2. A (nondeterministic) locally accepting 2-CFM $\mathcal{A}$ over $\{1, 2\}$ with set of synchronization messages $\{\diamondsuit, \square\}$ is illustrated in Fig. 8.1. As we

[1] Note that CM(Ag) does not impose any restriction on the state space, which can therefore produce non-recursive behavior.

deal with local acceptance, we depict a local final state by a second circle. Note that $L(\mathcal{A}) = \{\mathcal{M} \in \mathbb{MSC}(\{1,2\}) \mid \mathcal{M} \restriction 1 = (1!2)^n\,((1?2)(1!2))^n$ and $\mathcal{M} \restriction 2 = ((2?1)(2!1))^n\,(2?1)^n$ for some $n \in \mathbb{N}_{\geq 1}\}$ cannot be recognized by some CM with only one synchronization message, even if we allowed infinite local state spaces. Nevertheless, it can be recognized by some deterministic CFM. The verification of this is left to the reader as an exercise. In particular, the second component, $\mathcal{A}_2$, has to be modified accordingly.

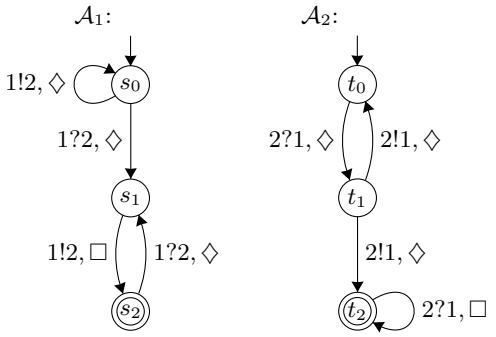


Fig. 8.1. A communicating finite-state machine

In [35, 45, 73], a run of a CM is defined on linearizations of MSCs rather than on MSCs, which reflects its operational behavior at the expense that several execution sequences might stand for one and the same run. As usual, such a view relies on the *global transition relation* of a CM, which, in turn, defers to the notion of a configuration. Let us be more precise and consider a CM $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$. By $S_\mathcal{A}$, we denote the set $\prod_{i\in Ag} S_i$ of *global states* of $\mathcal{A}$. 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 $\mathcal{A}$, $\Longrightarrow_\mathcal{A} \subseteq Conf_\mathcal{A} \times \Gamma \times \mathcal{D} \times Conf_\mathcal{A}$, is defined as follows:

- $((\overline{s}, \chi), i!j, m, (\overline{s}', \chi')) \in \Longrightarrow_\mathcal{A}$ if
  - $(\overline{s}[i], i!j, m, \overline{s}'[i]) \in \Delta_i$,
  - $\chi' = \chi[(i,j)/m \cdot \chi((i,j))]$, and
    (i.e., χ' maps (i,j) to $m \cdot \chi((i,j))$ and, otherwise, coincides with χ)
  - for all $k \in Ag \setminus \{i\}$, $\overline{s}[k] = \overline{s}'[k]$.
- $((\overline{s}, \chi), i?j, m, (\overline{s}', \chi')) \in \Longrightarrow_\mathcal{A}$ if there is a word $\underline{w} \in \mathcal{D}^*$ such that
  - $(\overline{s}[i], i?j, m, \overline{s}'[i]) \in \Delta_i$,
  - $\chi((j,i)) = \underline{w} \cdot m$,
  - $\chi' = \chi[(j,i)/\underline{w}]$, and
  - for all $k \in Ag \setminus \{i\}$, $\overline{s}[k] = \overline{s}'[k]$.

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* of $\mathcal{A}$, $\mathcal{A}$ defines in the canonical way a word language $L_{word}(\mathcal{A}) \subseteq \mathbb{W}(\Gamma)$ abstracting away synchronization messages. Namely, we define $\sigma_1 \dots \sigma_n \in \mathbb{W}(\Gamma)$ to be contained in $L_{word}(\mathcal{A})$ if there are messages $m_1, \dots, m_n \in \mathcal{D}$ and configurations $(\overline{s}_0, \chi_0), \dots, (\overline{s}_n, \chi_n) \in Conf_{\mathcal{A}}$ such that $(\overline{s}_0, \chi_0) = (\overline{s}^{in}, \chi_\varepsilon)$, $(\overline{s}_n, \chi_n) \in F \times \{\chi_\varepsilon\}$, and, for any $k \in \{1, \dots, n\}$, $((\overline{s}_{k-1}, \chi_{k-1}), \sigma_k, m_k, (\overline{s}_k, \chi_k)) \in \Longrightarrow_{\mathcal{A}}$. In particular, $L_{word}(\mathcal{A})$ uniquely determines an MSC language.

Lemma 8.3. *For any CM $\mathcal{A}$, we have $L_{word}(\mathcal{A}) = Lin(L(\mathcal{A}))$.*

Given configurations $(\overline{s}, \chi), (\overline{s}', \chi') \in Conf_{\mathcal{A}}$, we write $(\overline{s}, \chi) \rightsquigarrow_{\mathcal{A}} (\overline{s}', \chi')$ if, for some $\sigma \in \Gamma$ and $m \in \mathcal{D}$, we have $((\overline{s}, \chi), \sigma, m, (\overline{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}$. We call a configuration $(\overline{s}, \chi) \in Conf_{\mathcal{A}}$ *reachable* in $\mathcal{A}$ if $(\overline{s}^{in}, \chi_\varepsilon) \rightsquigarrow_{\mathcal{A}}^* (\overline{s}, \chi)$. Two possible steps of $\rightsquigarrow_{\mathcal{A}}$, the one concerning a send action, the other concerning a receive action, are illustrated in Fig. 8.2 and 8.3.

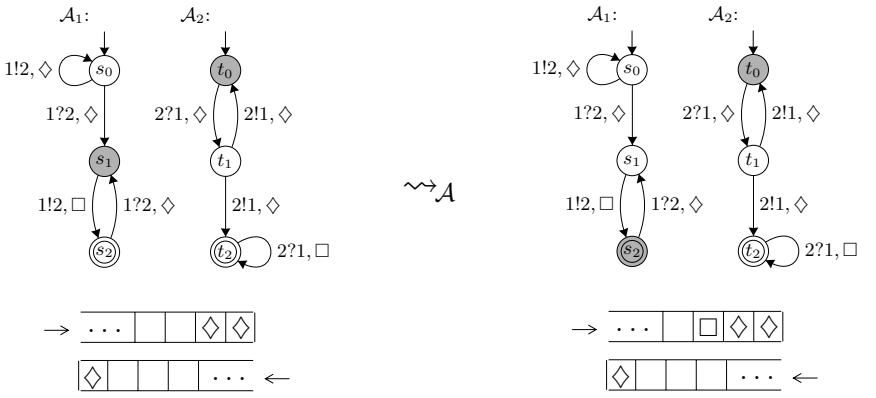


Fig. 8.2. The global transition relation of a CFM executing a send action

Exercise 8.4. Consider the configurations illustrated in Fig. 8.2 and 8.3. Which of these configurations are reachable in the CFM $\mathcal{A}$ from Fig. 8.1? Afterwards, determine the complete set of configurations that are reachable in $\mathcal{A}$.

Exercise 8.5. Show Lemma 8.3.

Note that, for any deterministic (finite) CM $\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, and any MSC $\mathcal{M}$, there is at most one run of $\mathcal{A}$ on $\mathcal{M}$. However, $\mathcal{A}$ can be extended towards a deterministic (finite, respectively) CM $\mathcal{A}' =$

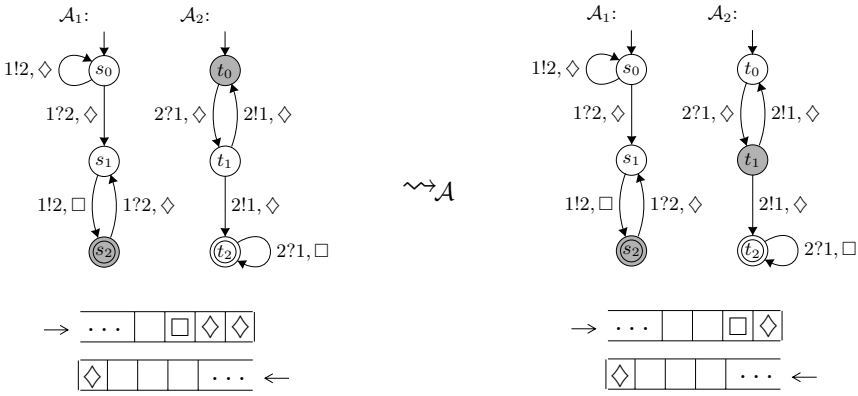


Fig. 8.3. The global transition relation of a CFM executing a receive action

$((\mathcal{A}'_i)_{i\in Ag}, \mathcal{D}', \overline{s}'_0, F')$, $\mathcal{A}'_i = (S'_i, \Delta'_i)$, such that both $L(\mathcal{A}') = L(\mathcal{A})$ and, for any MSC $\mathcal{M}$, there is *exactly* one run of $\mathcal{A}'$ on $\mathcal{M}$. To this aim, we extend the set of synchronization messages $\mathcal{D}$ by a message *fail*, i.e., $\mathcal{D}' = \mathcal{D} \cup \{fail\}$, and, for each $i \in Ag$, enrich S_i by a sink state $sink_i$, i.e., $S'_i = S_i \cup \{sink_i\}$. For any $i \in Ag$, Δ'_i already contains all the transitions from Δ_i. Moreover, transitions are added to Δ'_i according to the following procedure. For any $s \in S_i$ and $j \in Ag \setminus \{i\}$,

- if there is no $m \in \mathcal{D}$ and $s' \in S_i$ such that $(s, i!j, m, s') \in \Delta_i$, add a transition $(s, i!j, fail, sink_i)$, and
- add a transition $(sink_i, i!j, fail, sink_i)$.

For any $s \in S_i$, $j \in Ag \setminus \{i\}$, and $m \in \mathcal{D}$,

- if there is no $s' \in S_i$ such that $(s, i?j, m, s') \in \Delta_i$, add a transition $(s, i?j, m, sink_i)$,
- add a transition $(s, i?j, fail, sink_i)$, and
- add a transition $(sink_i, i?j, m', sink_i)$ 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 $F' = F$.

Thus, local states from S_i that lack an outgoing send transition labeled $i!j$ for some j, are enabled to send at least a message *fail* to j, though this is doomed to failure. Moreover, a missing receipt of a message m is made possible, even if it ends in a state $sink_i$, which does not contribute to an accepting run either. Thus, the above transformation affects neither the recognized language nor the property of being deterministic. But it enables the resulting automaton to execute a run on any given MSC. Those automata will be used to show in Chap. 9 that deterministic CFMs are strictly weaker than their nondeterministic counterpart.

8.2 Channel-Bounded and Deadlock-Free CMs

A natural restriction of CMs comes along with bounded channels whose capacity is a priori limited.

For $B \in \mathbb{N}_{\geq 1}$, a CM $\mathcal{A}$ 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 CMs, let $\forall\mathfrak{C}$ denote the set of $\forall$-bounded CMs $\mathcal{A}$ with $\mathcal{A} \in \mathfrak{C}$. The same principle as for universal boundedness applies to the *existential* one. In this sense, let $\exists$CFM denote the class of $\exists$-bounded CFMs. Note that the CFM from Fig. 8.1 is not even contained in $\exists$CFM.

Note that our definition of boundedness for CMs coincides with the one used in [56]. According to Henriksen et al., who use a slightly different notion of bounded CMs, we call a CM $\mathcal{A}$ *strongly* $\forall B$*-bounded* for some $B \geq 1$ if, for any $(i,j) \in Ch$ and any configuration $(\overline{s}, \chi)$ that is reachable in $\mathcal{A}$, $|\chi((i,j))| \leq B$. A CM $\mathcal{A}$ is called *strongly* $\forall$*-bounded* if it is strongly $\forall B$-bounded for some $B \geq 1$. Given a class $\mathfrak{C}$ of CMs, let $\forall\forall\mathfrak{C}$ denote the set of strongly $\forall$-bounded CMs $\mathcal{A}$ with $\mathcal{A} \in \mathfrak{C}$. Observe that a strongly $\forall$-bounded CM is $\forall$-bounded. Obviously, the 1-CFM from Fig. 8.4 is strongly $\forall$-bounded.

Besides determinism, the absence of *deadlocks* is a crucial aim when designing a distributed protocol. The study of realizability without deadlocks was conceived in [4] and then continued in [10] and [61]. While, to some extent, finite automata over words can be assumed to be free from deadlock states, which cannot contribute to an accepting run anymore, CFMs (and also asynchronous (cellular) automata) are more complicated in this regard: in general, deadlocks cannot be avoided. Even simple finite MSC languages are inherently *non-safe*. Moreover, it is undecidable if a CFM has a deadlock at all. Let us be more precise and suppose $\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \mathcal{D}, \overline{s}^{in}, F)$ to be a CM. We say that $(\overline{s}, \chi) \in Conf_{\mathcal{A}}$ is a *deadlock configuration* of $\mathcal{A}$ if there is no $(\overline{s}', \chi') \in F \times \{\chi_\varepsilon\}$ such that $(\overline{s}, \chi) \rightsquigarrow_{\mathcal{A}}^{*} (\overline{s}', \chi')$. We call $\mathcal{A}$ *safe* if there is no deadlock configuration reachable in $\mathcal{A}$. For a set $\mathfrak{C}$ of CMs, safe-$\mathfrak{C}$ will denote the class of safe CMs $\mathcal{A}$ with $\mathcal{A} \in \mathfrak{C}$.

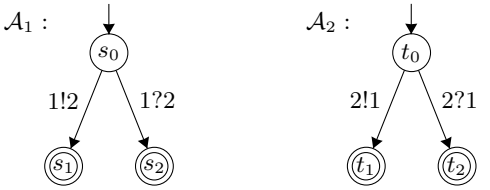


Fig. 8.4. A strongly $\forall$-bounded non-safe CFM

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

Example 8.6. Consider the non-safe CFM $\mathcal{A} \in$ 1-$\forall$CFM$_\ell$ over $\{1,2\}$ from Fig. 8.4 (say, with extra message $\diamondsuit$, which is omitted). It is non-safe, because the deadlock configuration $((s_1,t_1),((1,2)\mapsto\diamondsuit,(2,1)\mapsto\diamondsuit))$ is reachable in $\mathcal{A}$. It even holds that $L(\mathcal{A}) \notin \mathcal{L}$(safe-CM), i.e., the language of $\mathcal{A}$ cannot be recognized by some safe CM. The structural problem is that, in any implementation of $L(\mathcal{A})$ (provided there is only one global initial state), both agents 1 and 2 can independently decide to send a message, which inevitably leads into a deadlock configuration. Recall that those phenomena are known as *non-local-choice*.

Example 8.6 shows that $\mathcal{L}$(1-$\forall$CFM$_\ell$) and $\mathcal{L}$(safe-CM) are incomparable with respect to inclusion. In particular, the weakest model of a CFM is able to produce inherently non-safe MSC languages. However, it was shown in [37] that any local-choice HcMSC can be implemented by some safe locally accepting CFM.

Theorem 8.7 ([34, 37]).

$$\text{lc-}\mathcal{H}\text{c}\mathcal{MSC} \;\subseteq\; \mathcal{L}(\text{safe-CFM}_\ell)$$

To exemplify Theorem 8.7, consider the local-choice HMSC $G + G \cdot (G+H)^+$ with G and H taken from Fig. 7.6 on page 99. It is implemented by the safe locally accepting CFM that is illustrated in Fig. 8.5.

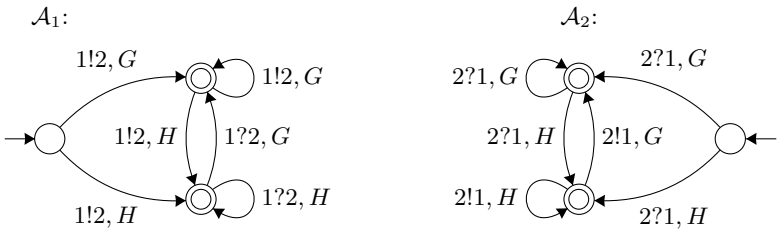


Fig. 8.5. A safe locally accepting CFM implementing a local-choice HMSC

Consider two variants of CMs. The first allows for accepting extended MSCs, say from $\langle \mathbb{MSC}, Q\rangle$ for some alphabet Q. Accordingly, for $i \in Ag$, the i-local transition relation of a CM is henceforth a subset of $S_i \times (\Gamma_i \times Q) \times \mathcal{D} \times S_i$. However, the type of an "action" $(\sigma, q) \in \Gamma \times Q$ still depends only on σ so that, in particular, a run may allow communicating events to have distinct additional labelings. Such an automaton will be used in Chap. 9 to characterize the language of some EMSO(Γ)-formula $\varphi(X_1,\ldots,X_n)$, which, as mentioned in Chap. 3, defines a subset of $\langle \mathbb{MSC}, \{0,1\}^n\rangle$.

A second variant of a CM allows us to specify different starting points of a run instead of one single global initial state. So let an *extended* CM be a CM $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, S^{in}, F)$ where, though, $S^{in} \subseteq \prod_{i\in Ag} S_i$ 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 CM case: an (accepting) run of $\mathcal{A}$ is an (accepting, respectively) run of one of the CMs $((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}_0, F)$ with $\overline{s}_0 \in S^{in}$. However, it turns out that several global initial states do not extend expressiveness.

Lemma 8.8. *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-CM iff it is the language of an extended locally accepting ($\forall$-bounded/finite/$\forall$-bounded and finite, respectively) N-CM.*

Proof. "Only if": Let $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be a CM. 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 $S_i^{\overline{s}}$ (we denote a copy of a local state $s \in S_i$ in $\mathcal{A}(\overline{s})$ by $s^{\overline{s}}$). The set of global final states is henceforth the cartesian product $\prod_{i\in Ag} \bigcup_{\overline{s}\in F} \{\overline{s}[i]^{\overline{s}}\}$. The resulting CM 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 $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, S^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be an extended CM where F is the cartesian product $\prod_{i\in Ag} F_i$ of sets $F_i \subseteq S_i$. Similarly to the "only if"-case, the basic idea is to create a copy $S_i^{\overline{s}_0} = S_i \times \{\overline{s}_0\}$ of S_i 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 simulates a run of $\mathcal{A}$ from $\overline{s}_0$ by henceforth allowing the system to enter no other copy than $S_i^{\overline{s}_0}$. In a global final state, it is then checked whether the other agents 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 some initial transitions $(\overline{s}^{in}[i], \sigma, m, (s, \overline{s}_0))$ if $(\overline{s}_0[i], \sigma, m, s)$ is some i-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 $i \in Ag$, either $\overline{s}[i] = \overline{s}^{in}[i]$ and $\overline{s}_0[i] \in F_i$ or $\overline{s}[i] \in F_i \times \{\overline{s}_0\}$. Again, neither the number of messages used nor boundedness or finiteness properties have changed. As F is finite, we also create a finite number of global final states only. □

8.3 Undecidability Results

We will now mention some undecidability results concerning emptiness, safety, and boundedness for CFMs. The following undecidability result is due to Brand and Zafiropulo [19].

Theorem 8.9 ([19]). *The following problem is undecidable:*

Input: $\mathcal{A} \in$ CFM.
Question: Is $L(\mathcal{A})$ empty?

Proof. We reduce the halting problem for Turing machines to the emptiness problem for CFMs (cf. Theorem 2.9). Say we are given a Turing machine $\mathcal{M} = (Q, \Sigma, \Delta, \Box, q_0, q_f)$. The CFM $\mathcal{A}$ simulating $\mathcal{M}$ will employ two agents. Roughly speaking, agent 1 will provide agent 2 with the current configuration by sending it as a word from left to right, letter by letter. The message alphabet of $\mathcal{A}$ is accordingly the set $\Sigma \times (Q \cup \{\Box\})$. However, for $a \in \Sigma$ and $q \in Q$, we will in the following write (a, q) as $a \leftarrow q$ and $(a, \Box)$ just as a. While reading a configuration step by step, agent 2 will locally modify it according to a given transition and then send the updated configuration letter by letter back to agent 1. Henceforth, the job of agent 1 is to send every letter it receives immediately back to agent 2, thus providing its opponent with the next configuration, which can be modified anew to obtain a successor configuration. Note that the length of configurations that are needed during a computation of a Turing machine cannot a priori be bounded by a natural number so that a simulating CFM will in general be unbounded (even existentially).

For a more detailed glimpse, consider Fig. 8.6, simulating the computation of a Turing machine that is depicted in Fig. 2.1 on page 15. First of all, agent 1, say being in state s_0 and moving to s_1, sends the letter $\Box$ to agent 2 together with the information that this is the letter that the Turing machine is currently reading in state q_0. Moving to state s_2, agent 1 sends # to agent 2, which is a separating symbol to indicate that the current configuration has been transmitted completely. From now on, agent 1 insistently sends any received message back to agent 2: being in state s_2, it can receive an arbitrary messages m and switch to state $s(m)$, from which it will reproduce the message m to make it available to agent 2 again.

Agent 2, starting from t_0, receives $\Box \leftarrow q_0$ and switches to a state that contains a suitable transition to be applied in q_0 while reading $\Box$. In the example, it decides in favor of the transition $(q_0, \Box, a, R, q_1)$, which means that the Turing machine is about to move to the right. Thus, the current configuration has to be extended on the right-hand side to obtain the corresponding successor configuration. It is up to agent 2 to provide agent 1 with the updated configuration. It therefore sends $\Box \leftarrow q_1$, which corresponds to what the Turing machine is about to read next, to agent 1, followed by the letter a, into which $\Box$ has changed according to the transition. Similarly to agent 1, agent 2 henceforth just sends the remainder of the current configuration back to agent 1 (in the example, however, it already receives #, which marks the end of a configuration, and sends it back).

Once more, agent 1 provides agent 2 with the updated configuration so that agent 2 may determine the third configuration of that computation: after having read #, the latter receives $\Box \leftarrow q_1$ to enter a state containing a suitable transition, say $(q_1, \Box, b, L, q_2)$. Thus, the Turing machine is changing $\Box$ into b and then moving left. Agent 2 therefore first sends b (instead of $\Box \leftarrow q_1$) back to agent 1, then receives a, which is what the Turing machine will read next, and accordingly sends $a \leftarrow q_2$ to its counterpart, agent 1.

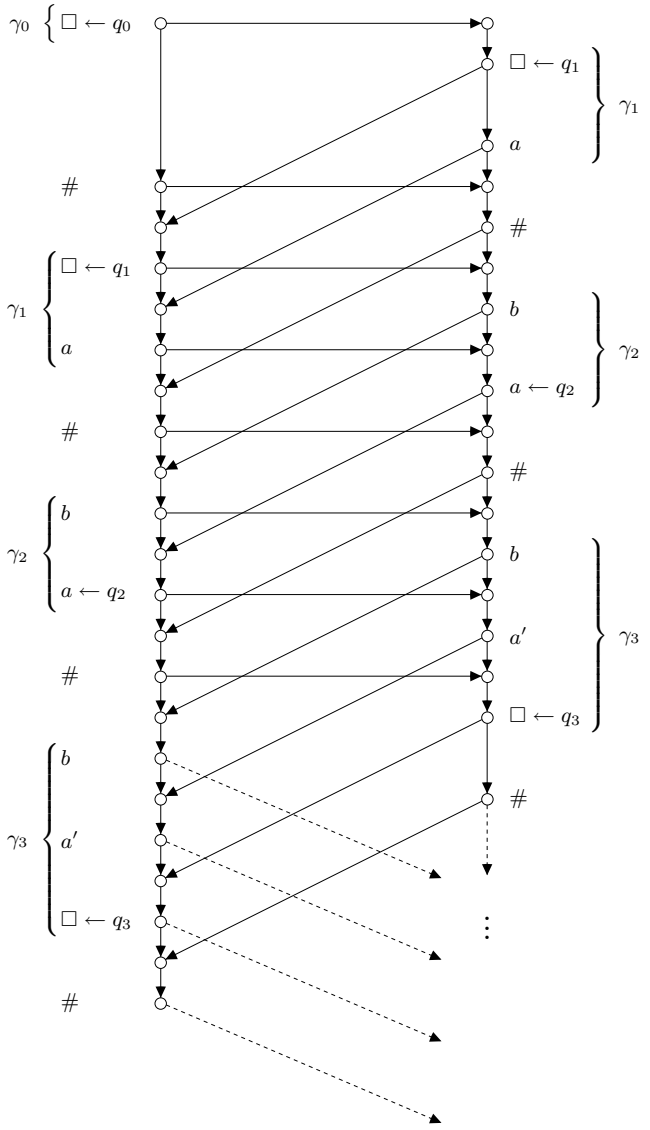


Fig. 8.6. An MSC simulating the computation of a Turing machine

Agent 2 now receives the end marker # and sends it back immediately, awaiting the updated configuration, which starts with a b. Reading the b, agent 2 will guess the "right" transition that has to be applied "somewhere" in the configuration. If we deal with a left transition or with a transition that does not make the Turing machine move, agent 2 may just wait for a symbol containing an arrow and a state of the Turing machine and to alter it correspondingly. Otherwise, concerning a right-moving transition, agent 2 has to guess in addition what the position right before the head is. For example, provided agent 2 decided in favor of (q_2, a, a', R, q_3) while reading the b, it would have to

- send $b \leftarrow q_3$ instead of just b, entering a state $t(a \leftarrow q_2)$,
- receive $a \leftarrow q_2$ (no other symbol can be received in state $t(a \leftarrow q_2)$), and
- send a' back to agent 1.

However, in the example, the left-moving transition (q_2, a, a', L, q_3) is applied so that agent 2

- sends b unchanged back to agent 1,
- detects (receives) $a \leftarrow q_2$,
- sends a' to agent 1 entering a state indicating that the symbol to be sent next has to be equipped with an arrow and state q_3, and
- receives # so that the symbol $\square \leftarrow q_3$ has to be inserted before returning #.

Recall that we are interested in a CFM whose language is empty iff the Turing machine at hand can reach a configuration featuring q_f. To this aim, we introduce two local final states s_f and t_f, one for agent 1 and one for agent 2, respectively. At any time, agent 1 may switch into s_f, in which arbitrary and arbitrarily many messages can be received to empty channel $(2, 1)$. Agent 2 is allowed to move into t_f and to empty the channel $(1, 2)$ as soon as it receives a letter $c \leftarrow q_f$ for some c.

As agent 2 modifies a configuration of the Turing machine locally and thereby needs to store only a subword of a configuration of constant length, finitely many states are sufficient so that, in fact, we deal with a CFM (that is even locally accepting). □

Observe that MSCs of the form depicted in Fig. 8.6 reflect the whole complexity of MSCs. Their structure will be used more often in this book. In particular, it constitutes a means to embed grids into MSCs and to transport results from the grid into the MSC setting.

Theorem 8.10. *The following problems are undecidable:*

(a) *Input:* $\mathcal{A} \in$ CFM. *Question: Is* $\mathcal{A}$ *safe?*
(b) *Input:* $\mathcal{A} \in$ CFM. *Question:* $\mathcal{A} \in \forall$CFM*?*
(c) *Input:* $\mathcal{A} \in$ CFM *and* $B \geq 1$. *Question: Is* $\mathcal{A}$ $\forall B$*-bounded?*

Note that the last two problems remain undecidable if we ask for existential boundedness.

Exercise 8.11. Prove Theorem 8.10. You may use Theorem 8.9.

Theorem 8.12. *The following problems are decidable:*

(a) *Input:* $\mathcal{A} \in$ safe-CFM *and* $B \geq 1$. *Question: Is* $\mathcal{A}$ $\forall B$*-bounded?*
(b) *Input:* $\mathcal{A} \in \forall\forall$CFM. *Question: Is* $L(\mathcal{A})$ *empty?*

Exercise 8.13. Prove Theorem 8.12.

8.4 Lossy Channel Systems

Under certain circumstances, communication through channels can be assumed to be unreliable so that messages, once they are sent, might get lost during their transmission, without notification of the receiving agent.

Definition 8.14 (Lossy Channel System). *A* lossy channel system *(LCS) (over Ag) is a structure* $((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$ *whose components correspond to those of a CFM, i.e.,*

- $\mathcal{D}$ *is a nonempty finite set of* synchronization messages*,*
- *for each* $i \in Ag$, $\mathcal{A}_i$ *is a pair* (S_i, Δ_i) *with* S_i *a nonempty finite set of* states *and* $\Delta_i \subseteq S_i \times \Gamma_i \times \mathcal{D} \times S_i$*,*
- $\overline{s}^{in} \in \prod_{i\in Ag} S_i$ *is the* global initial state*, and*
- $F \subseteq \prod_{i\in Ag} S_i$ *is the set of* global final states*.*

The behavior of an LCS is the same as that of a CFM except that it is henceforth considered relative to LMSCs: let $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be an LCS and $\mathcal{M} = (V, \lhd\, \lambda) \in \mathbb{LMSC}$. Again, a *run* of $\mathcal{A}$ on $\mathcal{M}$ is a pair (ρ, μ) of a mapping $\rho : V \to \bigcup_{i\in Ag} S_i$ with $\rho(u) \in S_{Ag(u)}$ for each $u \in V$ and a mapping $\mu : V \to \mathcal{D}$ such that

- for any $u, v \in V$ with $u \lhd_{\mathrm{c}} v$, $\mu(u) = \mu(v)$, and
- for any $u \in V$, $(source^{\overline{s}^{in}}_{(\mathcal{M},\rho)}(u)[Ag(u)], \lambda(u), \mu(u), \rho(u)) \in \Delta_{Ag(u)}$.

We call (ρ, μ) *accepting* if $final^{\overline{s}^{in}}_{(\mathcal{M},\rho)} \in F$ and denote by $L(\mathcal{A}) := \{\mathcal{M} \in \mathbb{LMSC} \mid$ there is an accepting run of $\mathcal{A}$ on $\mathcal{M}\}$ the *language* of $\mathcal{A}$. Finally, let $\mathcal{LCS}$ stand for $\{L \subseteq \mathbb{LMSC} \mid$ there is an LCS $\mathcal{A}$ such that $L(\mathcal{A}) = L\}$.

To adopt a rather operational view of an LCS, we will again employ a global transition relation. Consider an LCS $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$. We transfer the definitions of $Conf_{\mathcal{A}}$ and $\rightsquigarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times Conf_{\mathcal{A}}$ from (finite) CMs to the current setting and moreover let $\rightsquigarrow^{lcs}_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times Conf_{\mathcal{A}}$ be the union of $\rightsquigarrow_{\mathcal{A}}$ and $\overset{\ell}{\rightsquigarrow}_{\mathcal{A}}$ where $\overset{\ell}{\rightsquigarrow}_{\mathcal{A}} := \{((\overline{s}, \chi), (\overline{s}', \chi')) \in Conf_{\mathcal{A}} \times Conf_{\mathcal{A}} \mid \overline{s} = \overline{s}'$ and there are a channel $(i, j) \in Ch$, a message $m \in \mathcal{D}$, and words $\underline{w}_1, \underline{w}_2 \in \mathcal{D}^*$ such that $\chi((i, j)) = \underline{w}_1 m \underline{w}_2$ and $\chi' = \chi[(i, j)/\underline{w}_1\underline{w}_2]\}$.

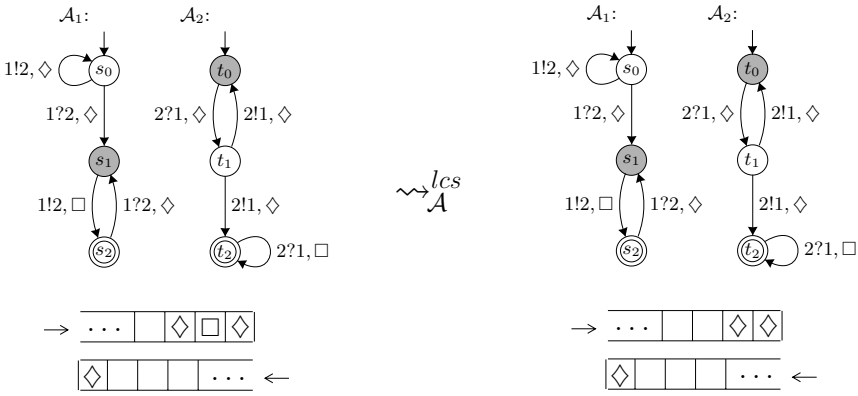


Fig. 8.7. The global transition relation of an LCS

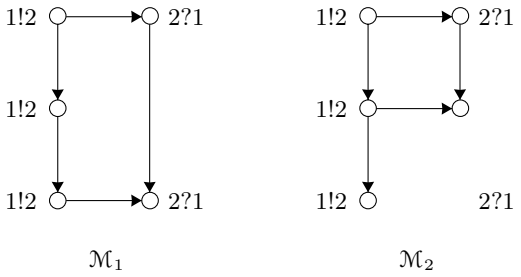


Fig. 8.8. Two LMSCs whose sets of linearizations are not disjoint

The latter relation, $\overset{\ell}{\rightsquigarrow}_{\mathcal{A}}$, reflects the operational *faulty* communication behavior of $\mathcal{A}$ and allows the system to remove messages from a queue. It is illustrated by Fig. 8.7. In contrast, the former relation, $\rightsquigarrow_{\mathcal{A}}$, is concerned with sending and receiving messages just as this was also the case when considering perfect channels. Thus, concerning $\rightsquigarrow_{\mathcal{A}}^{lcs} = \rightsquigarrow_{\mathcal{A}} \cup \overset{\ell}{\rightsquigarrow}_{\mathcal{A}}$, a configuration may henceforth lose, at any time of execution, an arbitrary message that is waiting in a channel for being received.

To establish a correspondence between a set of LMSCs and a word language is not straightforward, as two distinct LMSCs may share linearizations, i.e., a linearization is no longer exclusive to one single dag when going over from MSCs to LMSCs. Considering Fig. 8.8, for example, (1!2)(1!2)(1!2)(2?1)(2?1) is a linearization of both $\mathcal{M}_1$ and $\mathcal{M}_2$. Thus, we can no longer uniquely infer from a set of linearizations a corresponding collection of LMSCs. We may however establish the following characterization of the emptiness problem for LCSs in terms of their operational behavior.

Lemma 8.15. *Let* $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$ *be an LCS. We have* $L(\mathcal{A}) \neq \emptyset$ *iff* $((\overline{s}^{in}, \chi_\varepsilon), (\overline{s}, \chi_\varepsilon)) \in (\rightsquigarrow_{\mathcal{A}}^{lcs})^*$ *for some* $\overline{s} \in F$.

Exercise 8.16. Show Lemma 8.15.

Recall that assuming reliable channels leads to an undecidable emptiness problem (cf. Theorem 8.9). Surprisingly, emptiness is decidable for LCSs, though it has nonprimitive recursive complexity [86].

Theorem 8.17 ([1]). *The following problem is decidable:*

Input: LCS $\mathcal{A}$.
Question: Is $L(\mathcal{A})$ *empty?*

8.5 Non-Fifo Channel Systems

Though, for simplicity, we abstract in most sections from message contents, they can be easily included without major effort. In particular, a *communicating finite-state machine* over (Ag, Λ) (for a message alphabet Λ) is defined to be a structure $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$ such that $\mathcal{D}$, $\mathcal{A}_i = (S_i, \Delta_i)$, $\overline{s}^{in}$, and F are defined as in the case of a CFM over Ag except that $\Delta_i \subseteq S_i \times \Gamma_i(Ag, \Lambda) \times \mathcal{D} \times S_i$. Thus, the only difference between a CFM over Ag and a CFM over (Ag, Λ) is the transition relation, which is adapted either to actions from $\Gamma_i(Ag)$ or to actions from $\Gamma_i(Ag, \Lambda)$. Henceforth, $\mathcal{A}$ may run on (non-fifo) MSCs $\mathcal{M} = (V, \lhd, \lambda) \in \mathbb{MSC}_{\not\searrow}(Ag, \Lambda)$. Namely, a *run* of $\mathcal{A}$ on $\mathcal{M}$ is a pair (ρ, μ) of mappings $\rho : V \to \bigcup_{i\in Ag} S_i$ with $\rho(u) \in S_{Ag(u)}$ for each $u \in V$ and $\mu : V \to \mathcal{D}$ such that, as usual,

- for any $u, v \in V$ with $u \lhd_{\mathrm{c}} v$, $\mu(u) = \mu(v)$, and
- for any $u \in V$, $(source^{\overline{s}^{in}}_{(\mathcal{M},\rho)}(u)[Ag(u)], \lambda(u), \mu(u), \rho(u)) \in \Delta_{Ag(u)}$.

We call (ρ, μ) *accepting* if $final^{\overline{s}^{in}}_{(\mathcal{M},\rho)} \in F$. Thus, the definition of a run is the same as in the case of a CFM (or LCS) over Ag. Moreover, we get the notions of the (non-fifo) *language* $L(\mathcal{A})$ of $\mathcal{A}$ and of the corresponding language class $\mathcal{CFM}_{\not\searrow}(Ag, \Lambda)$ in the canonical manner.

Hence, the languages from $\mathcal{CFM}_{\not\searrow}(Ag, \Lambda)$ may contain MSCs that are non-fifo. Considering CFMs over (Ag, Λ) relative to $\mathbb{MSC}(Ag, \Lambda)$, however, we obtain the class $\mathcal{CFM}(Ag, \Lambda) \subseteq 2^{\mathbb{MSC}(Ag,\Lambda)}$, whose languages contain only fifo-dags.

So let us determine a global transition relation suitable for CFMs over (Ag, Λ) relative to $\mathbb{MSC}_{\not\searrow}(Ag, \Lambda)$. Rather than with respect to a channel (i, j), a configuration keeps track of the channel contents with respect to (i, j) *and* a message from Λ. Otherwise, the behavior of a CFM over (Ag, Λ) is basically the same as that of a CFM over Ag. In turn, a CFM over Ag might be seen as a special case of a CFM over (Ag, Λ) where Λ is a singleton set.

Let $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be a CFM over (Ag, Λ). The set of (non-fifo) *configurations* of $\mathcal{A}$, denoted by $Conf_{\mathcal{A}}$, is the cartesian product $S_{\mathcal{A}} \times \mathcal{C}_{\mathcal{A}}$ where $\mathcal{C}_{\mathcal{A}} := \{\chi \mid \chi : (Ch \times \Lambda) \to \mathcal{D}^*\}$ is the set of *channel contents* of $\mathcal{A}$. The (non-fifo) *global transition relation* of $\mathcal{A}$, $\Longrightarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times \Gamma(Ag, \Lambda) \times \mathcal{D} \times Conf_{\mathcal{A}}$, is given as follows:

- $((\overline{s}, \chi), i!^a j, m, (\overline{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}$ if
  - $(\overline{s}[i], i!^a j, m, \overline{s}'[i]) \in \Delta_i$,
  - $\chi' = \chi[((i,j),a)/m \cdot \chi((i,j),a)]$, and
  - for all $k \in Ag \setminus \{i\}$, $\overline{s}[k] = \overline{s}'[k]$.
- $((\overline{s}, \chi), i?^a j, m, (\overline{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}$ if there is a word $\underline{w} \in \mathcal{D}^*$ such that
  - $(\overline{s}[i], i?^a j, m, \overline{s}'[i]) \in \Delta_i$,
  - $\chi((j,i),a) = \underline{w} \cdot m$,
  - $\chi' = \chi[((j,i),a)/\underline{w}]$, and
  - for all $k \in Ag \setminus \{i\}$, $\overline{s}[k] = \overline{s}'[k]$.

The initial configuration, the set of final configurations, and the word language $L_{word}(\mathcal{A}) \subseteq \mathbb{W}(\Gamma(Ag, \Lambda))$ of $\mathcal{A}$ are obtained in the usual manner. The reader may verify the following correspondence:

Lemma 8.18. *For any CFM $\mathcal{A}$ over (Ag, Λ), we have $L_{word}(\mathcal{A}) = Lin(L(\mathcal{A}))$.*

Exercise 8.19. Determine a global transition relation that is suitable for CFMs over (Ag, Λ) relative to $\mathbb{MSC}(Ag, \Lambda)$ (thus, modeling a fifo behavior).

If not otherwise explicitly stated, we deal, further on, with CFMs and MSCs over Ag, abstracting from message contents.

8.6 CMs vs. Product MSC Languages

For the moment, let us put aside LCSs and non-fifo systems, and let us again address CMs. We will 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 8.20 ([4]).

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

Corollary 8.21.

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

Proof. "$\supseteq$": According to Lemma 8.8, a 1-CM can be transformed into an equivalent extended locally accepting 1-CM $((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, S^{in}, F)$, which then recognizes $\bigcup_{\overline{s}\in S^{in}} L(((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}, F))$, i.e., the finite union of languages that are each accepted by some locally accepting 1-CM. The assertion follows from Lemma 8.20.

"$\subseteq$": Similarly, any MSC language $L \in \mathcal{P}$ is the union of finitely many languages $L_1, \ldots, L_n \in \mathcal{P}^0$, which, according to Lemma 8.20, are recognized by locally accepting 1-CMs $\mathcal{A}^1, \ldots, \mathcal{A}^n$ (each employing, say, $\diamondsuit$ as synchronization message) with global initial states $\overline{s}_1, \ldots, \overline{s}_n$ and sets of global final states $F^1, \ldots, F^n$, respectively, where, for each $k \in \{1, \ldots, n\}$, $F^k = \prod_{i \in Ag} F_i^k$ for some sets $F_i^k \subseteq S_i^k$ (let S_i^k be the set of i-local states of $\mathcal{A}^k$). Without loss of generality, $\mathcal{A}^1, \ldots, \mathcal{A}^n$ have mutually distinct local state spaces. The extended locally accepting 1-CM recognizing L processwise merges the state spaces and transitions of $\mathcal{A}^1, \ldots, \mathcal{A}^n$, employs $\{\overline{s}_1, \ldots, \overline{s}_n\}$ being the set of global initial states, and, similarly to the proof of Lemma 8.8, $\prod_{i \in Ag} \bigcup_{k \in \{1,\ldots,n\}} F_i^k$ being the set of global final states. The assertion then follows from Lemma 8.8. □

8.7 CFMs vs. Regular MSC Languages

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

Theorem 8.22 ([45, 56]).

$$\mathcal{R} \;=\; \mathcal{L}(\forall\text{CFM}) \;=\; \mathcal{L}(\forall\forall\text{CFM})$$

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:

Lemma 8.23.

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

Proof. "$\supseteq$": According to Lemma 8.20, the language of a CFM $\mathcal{A} \in 1\text{-}\forall\text{CFM}_\ell$ is a weak product language, and, according to Theorem 8.22, $L_{word}(\mathcal{A}) = Lin(L(\mathcal{A}))$ constitutes a regular word language over Γ.

"$\subseteq$": Let $L \in \mathcal{R}\mathcal{P}^0$ and, for $i \in Ag$, let $\mathcal{A}_i = (S_i, \Delta_i, \overline{s}_i^{in}, F_i)$ be a finite automaton over Γ_i satisfying $L(\mathcal{A}_i) = L \restriction i := \{\mathcal{M} \restriction i \mid \mathcal{M} \in L\}$. Consider the CFM $\mathcal{A} = ((\mathcal{A}_i')_{i \in Ag}, \mathcal{D}, \overline{s}^{in}, F)$ with $\mathcal{D} = \{\diamondsuit\}$, $\overline{s}^{in} = (\overline{s}_i^{in})_{i \in Ag}$, $F = \prod_{i \in Ag} F_i$, and $\mathcal{A}_i' = (S_i, \Delta_i')$ where, for any $s, s' \in S_i$ and $\sigma \in \Gamma_i$, $(s, \sigma, \diamondsuit, s') \in \Delta_i'$ if $(s, \sigma, s') \in \Delta_i$. We claim that both $\mathcal{A} \in 1\text{-}\forall\text{CFM}_\ell$ and $L(\mathcal{A}) = L$. First, it is easy to see that $L \subseteq L(\mathcal{A})$. Now assume an MSC $\mathcal{M}$ to be contained in $L(\mathcal{A})$. For each $i \in Ag$, $\mathcal{M} \restriction i \in L(\mathcal{A}_i) = L \restriction i$ so that there is an MSC $\mathcal{M}' \in L$ with $\mathcal{M}' \restriction i = \mathcal{M} \restriction i$. From the definition of $\mathcal{P}^0$, it then immediately follows that $\mathcal{M}$ is contained in L, too. Clearly, $\mathcal{A}$ is finite, locally accepting, and $\forall$-bounded. □

We now show that Lemma 8.23 does not hold if we adopt the definition of strong universal boundedness proposed by Henriksen et al. (whereas Theorem 8.22 still holds).

Lemma 8.24.

$$\mathcal{RP}^0 \supsetneq \mathcal{L}(\text{1-}\forall\forall\text{CFM}_\ell)$$

Proof. It remains to show strictness. Let $L = \{\mathcal{M}_1\}^* \cup \{\mathcal{M}_2\}^*$ with $\mathcal{M}_1$ and $\mathcal{M}_2$ given by Fig. 8.9, and suppose there is a CFM $\mathcal{A} \in \text{1-CFM}_\ell$ with $L(\mathcal{A}) = L$. Then, for each natural number $n \geq 1$, the word

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

leads from the initial configuration of $\mathcal{A}$ via $\Longrightarrow_{\mathcal{A}}$ (neglecting the synchronization message) to some configuration $(\overline{s}, \chi)$ 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. □

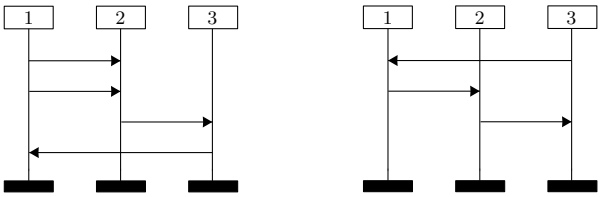


Fig. 8.9. Universal boundedness vs. strong boundedness

However, if we restrict to safely realizable MSC languages, Lemma 8.23 holds for both definitions of universal boundedness, as safely realizable languages can be recognized by CMs 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 8.25.

$$\mathcal{RP} = \mathcal{L}(\text{1-}\forall\text{CFM})$$

The proof of Corollary 8.25 is analogous to that of Corollary 8.21.

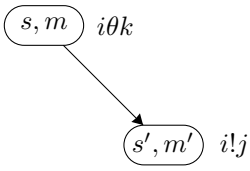
8.8 CFMs vs. ACAs and EMSO Logic

We now turn towards one of the central results of this book, which establishes the relationship between CFMs and EMSO logic over MSCs. In fact, any EMSO-definable MSC language is implementable as a CFM and, vice versa, any MSC language recognized by some CFM has an appropriate EMSO counterpart. However, according to Theorem 5.45 and Corollary 5.32, it suffices to establish the equivalence of CFMs and ACAs relative to $\mathbb{MSC}$.

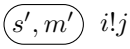
Lemma 8.26.

$$\mathcal{CFM} \;=\; \mathcal{ACA}_{\mathbb{MSC}}$$

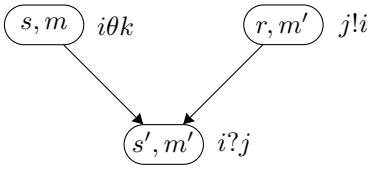
Proof. "$\subseteq$": We are given a CFM $((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, which shall be transformed into an ACA $\mathcal{A}' = (Q, \Delta, T, F')$ over $\widetilde{\Gamma}$ such that $L_{\mathbb{MSC}}(\mathcal{A}')$ equals $L(\mathcal{A})$. The set of states of $\mathcal{A}'$, Q, is the cartesian product of $\bigcup_{i\in Ag} S_i$ and $\mathcal{D}$, i.e., a state of $\mathcal{A}'$ incorporates both a local state of $\mathcal{A}$ and a synchronization message. This is because, unlike a CFM, an ACA is not capable of sending and receiving messages. The broadcast and reception of a message m in a CFM will therefore be mimicked by an ACA by assigning to communicating nodes only those states whose message components both equal m. More precisely, Δ contains a collection of transitions for any local transition from $\bigcup_{i\in Ag} \Delta_i$. Namely, a (sending) local CFM transition $(s, i!j, m', s') \in \Delta_i$ is represented by the collection of ACA transitions



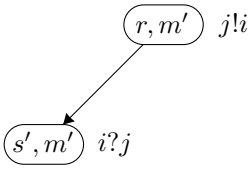
with $m \in \mathcal{D}$, $k \in Ag \setminus \{i\}$, and $\theta \in \{!, ?\}$. (Note that we might just have required that the upper left node is labeled with an action $\sigma \in \Gamma_i$ instead of $i\theta k$.) In addition, if $s = \overline{s}^{in}[i]$, Δ will also employ the following transition:



A (receiving) local CFM transition $(s, i?j, m', s') \in \Delta_i$ becomes



and, if $s = \overline{s}^{in}[i]$, it also becomes



where m ranges over $\mathcal{D}$, $k \in Ag \setminus \{i\}$, $\theta \in \{!, ?\}$, and $r \in S_j$.

Clearly, $T(\sigma, q) = \emptyset$ for any pair $(\sigma, q) \in \Gamma \times Q$. Finally, $\overline{q} \in (Q \uplus \{\imath\})^{Ag}$ is contained in F' if there is a global final state $\overline{s} \in F$ of $\mathcal{A}$ such that, for any $i \in Ag$, both $\overline{q}[i] = \imath$ implies $\overline{s}[i] = \overline{s}^{in}[i]$ and $\overline{q}[i] \in Q$ implies $\overline{q}[i] = (\overline{s}[i], m)$ for some $m \in \mathcal{D}$.

"$\supseteq$": Now suppose we are given an ACA $\mathcal{A} = (Q, \Delta, T, F)$ over $\widetilde{\Gamma}$. We build in the following a CFM $\mathcal{A}' = ((\mathcal{A}_i)_{i \in Ag}, \mathcal{D}, \overline{s}^{in}, F')$, $\mathcal{A}_i = (S_i, \Delta_i)$, such that $L(\mathcal{A}') = L_{\mathbb{MSC}}(\mathcal{A})$. Without loss of generality, we may assume that $\mathcal{A}$ is state separated and, moreover, that $\mathcal{A}$ is adjusted to $\mathbb{MSC}$, i.e., any transition from Δ matches one of the four transition types from the other proof direction above except that states are no longer pairs of a message and a state but elements from Q. We observe that a transition of $\mathcal{A}$ implies some global knowledge about the sending and the (distinct) receiving agent, i.e., about the states associated with sending and corresponding receiving nodes. Such global knowledge will be mimicked by the exchange of states through channels as follows: suppose Δ contains a transition $\{(i\theta k, q), (j!i, r)\} \longrightarrow (i?j, q')$. To emulate the transition in $\mathcal{A}'$, agent j will send r as a message to agent i to let him know that he is currently in state r. Receiving this message, agent i may then take the transition $(q, i?j, r, q')$, thus, making sure that his part matches a valid transition. Accordingly, we set $\mathcal{D}$ to be Q and, for any $i \in Ag$, S_i to be $Q \uplus \{\imath\}$.

For any $i \in Ag$, the set Δ_i will include a local transition $(q, i?j, r, q') \in S_i \times \Gamma_i^r \times \mathcal{D} \times S_i$ if

- Δ contains $\{(i\theta k, q), (j!i, r)\} \longrightarrow (i?j, q')$ for some $k \in Ag \setminus \{i\}$ and $\theta \in \{!, ?\}$ or
- Δ contains $\{(j!i, r)\} \longrightarrow (i?j, q')$ and $q = \imath$.

Moreover, for any $i \in Ag$, Δ_i will include a transition $(q, i!j, r, q') \in S_i \times \Gamma_i^s \times \mathcal{D} \times S_i$ if $r = q'$ and

- Δ contains $\{(i\theta k, q)\} \longrightarrow (i!j, q')$ for some $k \in Ag \setminus \{i\}$ and $\theta \in \{!, ?\}$ or
- Δ contains $\emptyset \longrightarrow (i!j, q')$ and $q = \imath$.

Finally, $\overline{s}^{in}$, the global initial state of $\mathcal{A}'$, is set to be $(\imath)_{i \in Ag}$ and F', the set of final states of $\mathcal{A}'$, is set to be F. We easily verify that then $\mathcal{A}'$ is equivalent to $\mathcal{A}$ relative to MSCs. □

Altogether, we may say that

> *Communicating finite-state machines are Asynchronous cellular automata relative to* $\mathbb{MSC}$.

Similarly, we can establish the expressive equivalence of LCSs and ACAs relative to LMSCs:

Lemma 8.27.

$$\mathcal{LCS} \;=\; \mathcal{ACA}_{\mathbb{LMSC}}$$

Thus, it is justified to say that

> *Lossy channel systems are*
> *Asynchronous cellular automata relative to* $\mathbb{LMSC}$.

Exercise 8.28. Show Lemma 8.27.

Exercise 8.29. Show that $\mathbb{MSC} \notin \mathcal{LCS}$.

Exercise 8.30. Show that $\mathcal{LCS} \subsetneqq \mathcal{ACAT}_{\mathbb{LMSC}}$.

From the characterizations of CFMs and LCSs in terms of ACAs, we obtain the following (un)decidability results for ACAs:

Corollary 8.31. *The following problem is undecidable:*

> *Input: ACA* $\mathcal{A}$ *over* $\widetilde{\Gamma}$.
> *Question: Is* $L_{\mathbb{MSC}}(\mathcal{A})$ *empty?*

Corollary 8.32. *The following problem is decidable:*

> *Input: ACA* $\mathcal{A}$ *over* $\widetilde{\Gamma}$.
> *Question: Is* $L_{\mathbb{LMSC}}(\mathcal{A})$ *empty?*

The following theorem, which might be considered to be the main result of this chapter, follows directly from Lemma 8.26, Theorem 5.45, and Corollary 5.32.

Theorem 8.33.

$$\mathcal{CFM} \;=\; \mathcal{EMSO}_{\mathbb{MSC}}$$

Regarding CFMs over (Ag, Λ) (with potential non-fifo behavior), we easily obtain a precise correspondence with their EMSO logic, too:

Theorem 8.34. *Let* Λ *be an alphabet.*

(a) $\mathcal{CFM}_{\searrow}(Ag, \Lambda) \;=\; \mathcal{EMSO}(\Gamma(Ag, \Lambda))_{\mathbb{MSC}_{\searrow}(Ag,\Lambda)}$
(b) $\mathcal{CFM}(Ag, \Lambda) \;=\; \mathcal{EMSO}(\Gamma(Ag, \Lambda))_{\mathbb{MSC}(Ag,\Lambda)}$

From Theorem 8.33, we can deduce that the satisfiability problem for EMSO sentences and the universality problem for Σ_2-sentences over $\mathbb{MSC}$ are undecidable.

Theorem 8.35. *The following two problems are undecidable:*

(a) *Input: Sentence* $\varphi \in \mathrm{EMSO}(\Gamma)$. *Question: Is* $L_{\mathbb{MSC}}(\varphi)$ *not empty?*
(b) *Input: Sentence* $\varphi \in \Sigma_2(\Gamma)$. *Question:* $L_{\mathbb{MSC}}(\varphi) = \mathbb{MSC}$*?*

Proof. Using Theorem 8.9 and Theorem 8.33, we get Theorem 8.35(a). Theorem 8.35(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. □

Exercise 8.36. Show that universality for EMSO sentences over $\mathbb{MSC}$ is undecidable.

We have (implicitly) shown that one can effectively transform a CFM into an EMSO sentence that is equivalent to the CFM over MSCs. Let us however give an explicit transformation from CFMs directly into an EMSO formula using a slightly different technique than we employed proving Lemma 5.38. It allows such a transformation without knowing the structure of local automata in detail, which might be given just by an EMSO sentence interpreted over words.

So let $N \geq 1$ and let $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \{1,\dots,N\}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be a CFM. We assume $F \neq \emptyset$. Note that $\mathcal{A}_i$, once equipped with an initial state $s \in S_i$ and a set of final states $F_i \subseteq S_i$, can be considered as a finite word automaton over $\Gamma_i \times \{1,\dots,N\}$ generating a regular word language. In case that $N = 1$, we can even understand $\mathcal{A}_i$ to recognize a word language over Γ_i ignoring the respective message component in the transition relation Δ_i.

Our aim is to exhibit an EMSO(Γ)-sentence Ψ such that $L_{\mathbb{MSC}}(\Psi) = L(\mathcal{A})$. Recall that the class of word languages that are $\mathrm{EMSO}_{\mathbb{W}}$-definable coincides with the class of regular word languages. Clearly, the language of $\mathcal{A}$ equals $\bigcup_{\overline{s}\in F} L(((\mathcal{A}_i)_{i\in Ag}, \{1,\dots,N\}, \overline{s}^{in}, \{\overline{s}\}))$. So let, for each global final state $\overline{s} \in F$ and each agent $i \in Ag$,

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

be an EMSO($\Gamma_i \times \{1,\dots,N\}$)-sentence with first-order kernel $\psi^{\overline{s},i}$ such that $L_{\mathbb{W}(\Gamma_i\times\{1,\dots,N\})}(\varphi^{\overline{s},i})$ is the language of the finite automaton $\mathcal{A}_i$ with initial state $\overline{s}^{in}[i]$ and set of final states $\{\overline{s}[i]\}$. 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 suitably labeled and, on the other hand, meets the restrictions imposed by the formulas $\varphi^{\overline{s},i}$ for at least one final state $\overline{s} \in F$. It is given by

$$\begin{aligned}\Psi = \exists X_1 \dots \exists X_N\ \exists \overline{X}\\ \Big(\ &\mathrm{partition}(X_1,\dots,X_N)\\ \wedge\ &\mathrm{consistent}(X_1,\dots,X_N)\\ \wedge\ &\textstyle\bigvee_{\overline{s}\in F} \bigwedge_{i\in Ag} \mathrm{agent}^{\overline{s},i}(X_1,\dots,X_N,X_1^{\overline{s},i},\dots,X_{n_{\overline{s},i}}^{\overline{s},i})\Big)\end{aligned}$$

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

The formula partition$(X_1,\dots,X_N)$ ensures that the variables $X_1,\dots,X_N$ define a mapping from the set of events of an MSC to the set of synchronization messages $\{1,\dots,N\}$.

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 with respect to the alphabet of synchronization messages:

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

For $\overline{s} \in F$ and $i \in Ag$, the formula agent$^{\overline{s},i}(X_1, \ldots, X_N, X_1^{\overline{s},i}, \ldots, X_{n_{\overline{s},i}}^{\overline{s},i})$ ensures that the total order that agent i induces with respect to an MSC corresponds to what the word automaton $\mathcal{A}_i$ with initial state $\overline{s}^{in}[i]$ and set of final states $\{\overline{s}[i]\}$ recognizes. It is defined to be the projection $\|\psi^{\overline{s},i}\|_i$ of the formula $\psi^{\overline{s},i}$ on i and derived inductively. In particular (recall that $\psi^{\overline{s},i}$ is the *first-order* kernel of a formula interpreted over words $(V, \lhd, \lambda) \in \mathbb{W}(\Gamma_i \times \{1, \ldots, N\})$ with $\lambda : V \to \Gamma_i \times \{1, \ldots, N\}$),

- $\|\lambda(x) = (\sigma, n)\|_i = (\lambda(x) = \sigma) \wedge x \in X_n$,
- $\|x \in X\|_i = x \in X$,
- $\|x \lhd y\|_i = x \lhd_i y$,
- $\|\exists x \varphi\|_i = \exists x (Ag(x) = i \wedge \|\varphi\|_i)$, and
- $\|\forall x \varphi\|_i = \forall x (Ag(x) = i \rightarrow \|\varphi\|_i)$,

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

8.9 CFMs vs. Graph Acceptors

We have implicitly shown that $\mathcal{CFM}$ and $\mathcal{GA}_{\mathbb{MSC}}$ coincide. As in most settings, we cannot generally remove occurrence constraints from graph acceptors over MSCs. However, we may restrict to certain MSC languages:

Lemma 8.37. *For any language $L \subseteq \mathbb{MSC}$ including $\mathbf{1}_{\mathbb{MSC}}$, we have*

$$L \in \mathcal{L}(\text{CFM}_\ell) \quad \textit{implies} \quad L \in 1\text{-}\mathcal{GA}^-_{\mathbb{MSC}}$$

Proof. We follow the proofs of Theorem 5.49 and Lemma 8.26. So let $\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$ and $F = \prod_{i \in Ag} F_i$, be a locally accepting CFM such that a state $s \in F_i$ is exclusive to a Γ_i-maximal event. We easily build a graph acceptor $\mathcal{B} = (Q, R, \mathcal{S}, Occ)$ over Γ such that $Q = (\bigcup_{i \in Ag} S_i) \times \mathcal{D}$, $R = 1$, $L_{\mathbb{MSC}}(\mathcal{B}) = L(\mathcal{A})$, and Occ is of the form

$$\bigwedge_{i \in Ag} \Big(\bigvee_{q \in F_i \times \mathcal{D},\ \mathcal{H} \in \mathcal{S}_q} \mathcal{H} \geq 1 \vee \bigwedge_{\overline{s}^{in}[i] \in F_i,\ \sigma \in \Gamma_i,\ \mathcal{H} \in \mathcal{S}_\sigma} \neg (\mathcal{H} \geq 1) \Big)$$

Provided $\overline{s}^{in} \in F$, Occ can be assumed to be true if we remove any sphere from $\mathcal{S}$ that, with respect to some $i \in Ag$, has a maximal sphere center that is not labeled with a state from F_i. □

Altogether, we observe the following correspondence:

> *Messages and local states in CFMs correspond to control states of graph acceptors, global final states correspond to occurrence conditions.*

Note that $\mathcal{EP}^0 \subseteq \mathcal{L}(\mathrm{CFM}_\ell)$ so that, in particular, the above result holds for weak EMSO-definable product MSC languages. The class $\mathcal{L}(\mathrm{CFM}_\ell)$ is not only interesting because of Lemma 8.37. It is the basis for an algorithm that, given a *locally cooperative* high-level MSC, yields a corresponding locally accepting CFM [37] (cf. also Theorem 8.7). Furthermore, if we restrict to connected MSCs that share at least one agent, then $\mathcal{L}(\mathrm{CFM}_\ell)$ and $\mathcal{EMSO}_{\mathbb{MSC}}$ coincide.

Lemma 8.38. *Let L be an MSC language. If there is $p \in Ag$ such that, for any $\mathcal{M} \in L \setminus \{\mathbf{1}_{\mathbb{MSC}}\}$, both $p \in Ag(\mathcal{M})$ and $\mathcal{M}$ is connected, then*

$$L \in \mathcal{L}(\mathrm{CFM}_\ell) \quad \textit{iff} \quad L \in \mathcal{L}(\mathrm{CFM})$$

Proof. Let $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, be a CFM and $p \in Ag$ such that, for any $\mathcal{M} \in L(\mathcal{A}) \setminus \{\mathbf{1}_{\mathbb{MSC}}\}$, $p \in Ag(\mathcal{M})$ and $\mathrm{CG}(\mathcal{M})$ is connected. We are interested in a locally accepting CFM $\mathcal{A}' = ((\mathcal{A}'_i)_{i\in Ag}, \mathcal{D}', \overline{s}'_0, F')$, $\mathcal{A}'_i = (S'_i, \Delta'_i)$, such that $L(\mathcal{A}') = L(\mathcal{A})$. Let us first describe the idea behind the construction of $\mathcal{A}'$. Each agent will basically guess both a global final state and a connected communication graph. In addition, a component i of a global final state might be $-$, which shall indicate that i is not expected to move at all. In that case, i 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 agent i, the set of previous communication partners of i must coincide with the set of i's direct neighbors in the communication graph.

Let $S^-_{\mathcal{A}}$ abbreviate $\prod_{i\in Ag}(S_i \cup \{-\})$ and let F^- be the set of tuples $\overline{s} \in S^-_{\mathcal{A}}$ for which there is $\overline{f} \in F$ such that $\overline{s}$ coincides with $\overline{f}$ in at least two components and, for all other components $i \in Ag$, we have both $\overline{s}[i] = -$ and $\overline{f}[i] = \overline{s}^{in}[i]$. Then, $\mathcal{A}'$ is given in detail as follows:

- for any $i \in Ag$, $S'_i = \left(S_i \times F^- \times (\prod_{i\in Ag} 2^{Ag}) \times 2^{Ag}\right) \cup \{\iota_i\}$
  (apart from the i-local initial state ι_i, the first component of an i-local state simulates $\mathcal{A}$, while the second component holds a guessed global final state, which, once chosen, cannot be changed anymore; the symbol "$-$" shall indicate that the respective agent is not expected to move at all; the third and fourth component specify further communication obligations and the set of former communication partners, respectively; hereby,

a communication graph $(Ag(\mathcal{M}), Arcs)$ of some MSC $\mathcal{M}$ will be represented by its undirected variant, namely as a tuple $\bar{\vartheta} \in \prod_{i \in Ag} 2^{Ag}$ where, for any $i, j \in Ag$, $j \in \bar{\vartheta}[i]$ iff both $\{i, j\} \subseteq Ag(\mathcal{M})$ and $(i, j) \in Arcs$ or $(j, i) \in Arcs$),

- $\mathcal{D}' = \mathcal{D} \times F^- \times \prod_{i \in Ag} 2^{Ag}$
  (again, the first component of a message aims at simulating the original CFM, 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 $\theta \in \{!, ?\}$, $(\iota_i, i\theta j, (d, \bar{d}, \bar{\vartheta}_1), (s', \overline{s}', \bar{\vartheta}', \vartheta')) \in \Delta'_i$ if
  - $(\overline{s}^{in}[i], i\theta j, d, s') \in \Delta_i$
    (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 $\bar{\vartheta}'[p] \neq \emptyset$ and, for any $k \in Ag$, $\overline{s}'[k] = -$ iff $\bar{\vartheta}'[k] = \emptyset$
    (agent i assumes both a global final state $\overline{s}'$ and a communication structure $\bar{\vartheta}'$ and propagates $\overline{s}'$ along $\bar{\vartheta}'$ to guarantee an agreement on $\overline{s}'$),
  - $(\bar{d}, \bar{\vartheta}_1) = (\overline{s}', \bar{\vartheta}')$,
  - $\vartheta' = \{j\}$
    (the set of communication partners of i is initialized to $\{j\}$),
  - $j \in \bar{\vartheta}'[i]$ and, consequently, $i \in \bar{\vartheta}'[j]$
    (however, the transition can only be taken if this is intended by $\bar{\vartheta}'$),
- for $\theta \in \{!, ?\}$, $((s, \overline{s}, \bar{\vartheta}, \vartheta), i\theta j, (d, \bar{d}, \bar{\vartheta}_1), (s', \overline{s}', \bar{\vartheta}', \vartheta')) \in \Delta'_i$ if
  - $(s, i\theta j, d, s') \in \Delta_i$
    (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 i's communication partner j, respectively),
  - $\vartheta' = \vartheta \cup \{j\}$
    (j is added to the set of i's communication partners so far),
  - $j \in \bar{\vartheta}[i]$ and, consequently, $i \in \bar{\vartheta}[j]$
    (j is only a valid communication partner of j if this is provided by the communication structure that is represented by $\bar{\vartheta}$),
- $\overline{s}'_0 = (\iota_i)_{i \in Ag}$
  (at the very beginning, no agent has made a guess about global final states and the communication structure), and
- $F' = \prod_{i \in Ag} F'_i$ where, for any $i \in Ag$, F'_i contains any tuple $(s, \overline{s}, \bar{\vartheta}, \vartheta)$ such that both $s = \overline{s}[i]$ and $\vartheta = \bar{\vartheta}[i]$, and it contains ι_i iff $i \neq p$ or $\mathbf{1}_{\mathbb{MSC}} \in L$
  (unless it did not make any move, agent i has to agree with the global final state and, according to $\bar{\vartheta}$, should have communicated with all the agents it was supposed to do). □

Exercise 8.39. Show that, if $|Ag| \leq 3$, then $\mathcal{L}(\text{CFM}_\ell) = \mathcal{L}(\text{CFM})$.

Lemma 8.40.

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

Proof. Due to Lemma 8.37 and Exercise 8.39, we cannot expect to find a corresponding language that contains $\mathbf{1}_{\mathbb{MSC}}$ and is defined over only three agents. However, it is easy to see that the language $\{G, I, \mathbf{1}_{\mathbb{MSC}}\}$ with G and I taken from Fig. 7.6 on page 99 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 3.23. □

Theorem 8.41. *In general, $\mathcal{MSO}[\leq]_{\mathbb{MSC}(Ag,\Lambda)}$ and $\mathcal{EMSO}_{\mathbb{MSC}(Ag,\Lambda)}$ are incomparable with respect to inclusion.*

Proof. We provide an MSC language that is $\text{EMSO}_{\mathbb{MSC}}$- but not $\text{MSO}[\leq]_{\mathbb{MSC}}$-definable (cf. [56]). It does not rely on message contents, which are therefore omitted. So consider the MSC language L that contains the MSCs $\mathcal{M}(m,n)$, $m,n \in \mathbb{N}$, of the form depicted in Fig. 8.10. Note that, for any $(V, \lhd, \lambda) \in L$, $\leq$ is a total order. It is easy to find a CFM recognizing L so that L is $\text{EMSO}_{\mathbb{MSC}}$-definable. This is left to the reader as an exercise. Now suppose L to be $\text{MSO}[\leq]_{\mathbb{MSC}}$-definable, too. Then, $Lin(L)$ is $\text{MSO}(\Gamma)[\leq]_{Lin(T)}$-definable where T is the set of totally ordered MSCs. Thus, there is, according to Theorem 4.10, a word language $L' \in \mathcal{FA}(\Gamma)$ such that $L' \cap Lin(T) = Lin(L)$. 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 Fig. 8.10. 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 totally ordered MSC, is not in accord with Fig. 8.10 and, therefore, contradicts our premise.

Conversely, set Ag 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 3.28 and recall that any picture from L can be (uniquely) partitioned into pictures $\mathcal{G}$, $\mathcal{C}$, and $\mathcal{H}$ such that the sets of different column labelings of $\mathcal{G}$ and $\mathcal{H}$ coincide. The unique partition of the picture from Fig. 3.6 is illustrated in Fig. 3.7 on page 32. Such a picture can be encoded as an MSC over (Ag, Λ) as follows. In the initialization phase, agent 1 sends arbitrarily many messages to agent 2. Thereupon, any message received is acknowledged immediately until one of the agents stops acknowledging.

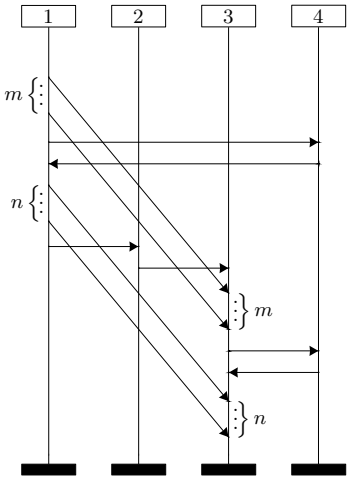


Fig. 8.10. An MSC language that is not MSO$[\leq]_{\mathbb{MSC}}$-definable

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 agent 1 and so on. In this sense, the picture from Fig. 3.6 gives rise to the MSC from Fig. 8.11. Note that only a send event corresponds to some point in the picture at hand. We omit here a formal definition of such a folding, as, though slightly modified, it is given in the following chapter.

Now suppose the MSC language L', which shall consist of the empty MSC $\mathbf{1}_{\mathbb{MSC}}$ and all foldings of pictures from L, to be the language of some CFM. Then, due to Lemma 8.37 and Exercise 8.39 (both adapted to MSCs over (Ag, Λ)), there is a graph acceptor $\mathcal{B} = (Q, 1, \delta, Occ)$ over $\Gamma(Ag, \Lambda)$ without occurrence constraints such that $L_{\mathbb{MSC}(Ag,\Lambda)}(\mathcal{B}) = L'$. An accepting run of $\mathcal{B}$, as argued in the proof of Theorem 3.28, 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 MSO$[\leq]_{\mathbb{MSC}(Ag,\Lambda)}$-definable. First of all, the set of all foldings of pictures from L is MSO$[\leq]_{\mathbb{MSC}(Ag,\Lambda)}$-definable. The corresponding formula basically claims the existence of a chain that starts at the

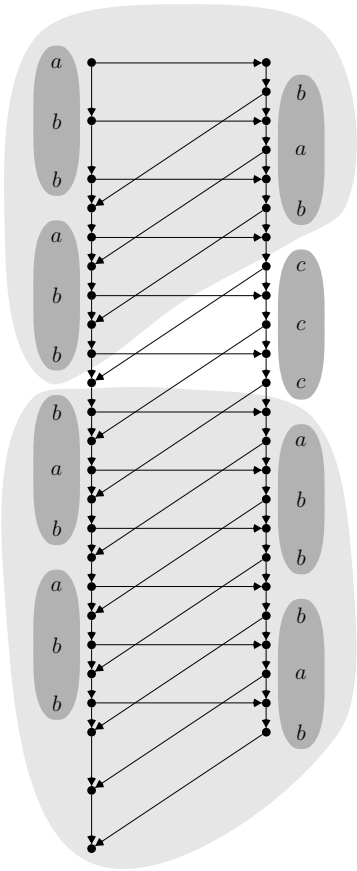


Fig. 8.11. Folding a picture

minimal event and alternates between the agents 1 and 2. Moreover, both $\leq_1$ and $\leq_2$, which are not to be confused with the orderings induced by the agents but correspond to walking in the grid from top to bottom and from left to right, respectively, are MSO[$\leq$]-definable with respect to MSCs. 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 vertices 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 next chapter (Corollary 9.9) that even the classes $\mathcal{MSO}[\leq]_{\mathbb{MSC}(Ag)}$ and $\mathcal{EMSO}_{\mathbb{MSC}(Ag)}$ (thus, without any message contents) are incomparable with respect to inclusion.

8.10 CFMs 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 deceptive:

Lemma 8.42. *We have the following strict inclusions:*

(a) $\mathcal{L}(\text{1-CFM}_\ell) \subsetneqq \mathcal{EP}^0$,
(b) $\mathcal{L}(\text{1-CFM}) \subsetneqq \mathcal{EP}$.

Proof. Inclusion of (a) follows from Lemma 8.20 and Theorem 8.33. Inclusion of (b) then proceeds as the proof for Corollary 8.21. Let us turn towards strictness. For natural numbers $m, n \geq 1$, let the MSC $\mathcal{M}(m,n)$ over $\{1,2\}$ be given by its projections $\mathcal{M}(m,n) \restriction 1 = (1!2)^m\,((1?2)(1!2))^n$ and $\mathcal{M}(m,n) \restriction 2 = ((2?1)(2!1))^n\,(2?1)^m$. The MSC $\mathcal{M}(3,2)$ is depicted in Fig. 8.12. Now consider the EMSO-definable MSC language $L = \{\mathcal{M}(n,n) \mid n \geq 1\}$, which is recognized by the locally accepting 2-CFM from Fig. 8.1 on page 119 with the set of synchronization messages $\{\diamondsuit, \square\}$. We easily verify that L is a weak product MSC language. However, L is not contained in $\mathcal{L}(\text{1-CFM})$. Because suppose there is a 1-CFM $\mathcal{A} = ((\mathcal{A}_i)_{i\in\{1,2\}}, \mathcal{D}, \overline{s}^{in}, F)$ with $L(\mathcal{A}) = L$. As $\mathcal{A}$ is finite, there is a natural number $n \geq 1$ and an accepting run of $\mathcal{A}$ on $\mathcal{M}(n,n)$ such that component $\mathcal{A}_1$, when reading the first n letters 1!2 of $\mathcal{M}(n,n) \restriction 1$, goes through a cycle, say of length k (≥ 1), and component $\mathcal{A}_2$, when reading the last n letters 2?1 of $\mathcal{M}(n,n) \restriction 2$, goes through another cycle, say of length l (≥ 1). (We just have to choose n large enough.) But then there is also an accepting run of $\mathcal{A}$ on $\mathcal{M}(n + (k \cdot l), n) \notin L$, which contradicts the premise. □

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

Lemma 8.43. *Let L be a finitely generated or universally bounded MSC language.*

(a) $L \in \mathcal{EP}^0$ *iff* $L \in \mathcal{L}(\text{1-CFM}_\ell)$,
(b) $L \in \mathcal{EP}$ *iff* $L \in \mathcal{L}(\text{1-CFM})$.

Proof. In the 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 [73], it follows that $L \in \mathcal{L}(\text{1-CFM}_\ell)$. Now assume L to be bounded. As L is already EMSO-definable, it is even regular [45]. The result then follows from Lemma 8.23. Proving (b) proceeds as the proof of Corollary 8.21. □

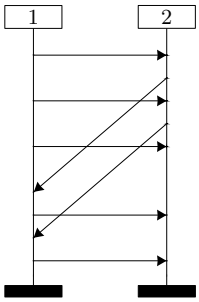


Fig. 8.12. The MSC $\mathcal{M}(3,2)$

8.11 The Complete Hierarchy

So far, we investigated the expressiveness of CMs with respect to some language classes proposed in Chap. 7. So let us summarize those results towards a hierarchy of MSC languages.

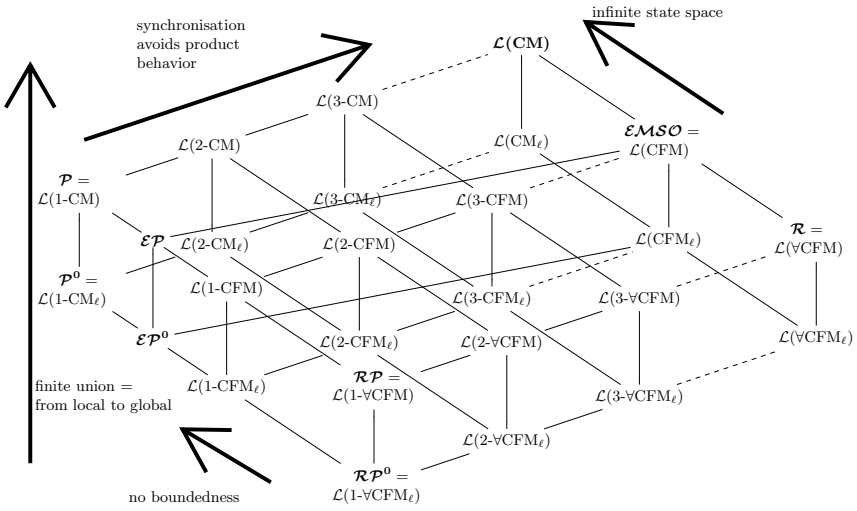


Fig. 8.13. A hierarchy of MSC languages

Theorem 8.44. *The classes of MSC languages proposed so far draw the picture given by Fig. 8.13.*

Theorem 8.44 follows from the results of the preceding sections as well as Lemma 8.45 and Lemma 8.47.

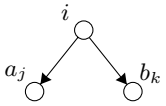


Fig. 8.14. Trace $\mathcal{T}(i,j,k)$ over $\widetilde{\Sigma}_{\widehat{N}}$

Lemma 8.45. *For each* $N \geq 1$, $\mathcal{L}((N+1)\text{-}\forall\text{CFM}_\ell) \setminus \mathcal{L}(N\text{-CM}) \neq \emptyset$.

Proof. We shift the proof into the setting of traces, i.e., we specify a language from $\mathcal{L}((N+1)\text{-}\forall\text{CFM}_\ell)$ that turns out to be contained in $\mathcal{L}(N\text{-CM})$ 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 8.46. Let $\widehat{N} \geq 1$ be a natural number, 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 $\widetilde{\Sigma}_{\widehat{N}}$ 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 (M$^-$)-trace $\mathcal{T}(i,j,k)$ over $\widetilde{\Sigma}_{\widehat{N}}$ to be the one illustrated in Fig. 8.14 and the trace language $\mathcal{T}$ to be $\{\mathcal{T}(i,j,j) \mid i \in \{1, \ldots, \widehat{N}\} \text{ and } j \in \{1, \ldots, \widehat{N}+1\}\}^*$. There is no product automaton $\mathcal{A}$ over $\widetilde{\Sigma}_{\widehat{N}}$ such that $L^-(\mathcal{A}) = \mathcal{T}$ (even if we allow infinite local state spaces).

Proof of Claim 8.46. 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 $\widetilde{\Sigma}_{\widehat{N}}$, $\mathcal{A}_k = (S_k, \Delta_k)$, with $L^-(\mathcal{A}) = \mathcal{T}$. For $k = 1, 2$, we define 2^{S_k}-labeled *trees* $t_k = (Dom_k, val_k)$ where $Dom_k \subseteq \mathbb{W}(\{1, \ldots, \widehat{N}+1\})$, the set of *nodes*, is the least set satisfying the following:

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

The *valuation function* val_k is given by

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

Let $i \geq 1$ be a natural number and let furthermore $\underline{u} = \underline{u}(1)\underline{u}(2)\ldots\underline{u}(2i) \in Dom_1$ and $\underline{v} = \underline{v}(1)\underline{v}(2)\ldots\underline{v}(2i) \in Dom_2$ be nodes of t_1 and t_2, respectively, such that $val_1(\underline{u}) \neq \emptyset$ and $val_2(\underline{v}) \neq \emptyset$. While $\underline{u}$ stands for a set of computations of $\mathcal{A}_1$, each ending in a state from $val_1(\underline{u})$, $\underline{v}$ 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 $(\underline{u}, \underline{v})$

represents a set of runs of $\mathcal{A}$ on $\mathcal{T}(\underline{u}(1),\underline{u}(2),\underline{v}(2))\cdot\ldots\cdot\mathcal{T}(\underline{u}(2i-1),\underline{u}(2i),\underline{v}(2i))$ if, for each $j \in \{1,\ldots,i\}$, $\underline{u}(2j-1) = \underline{v}(2j-1)$. (Of course, only those pairs $(\underline{u},\underline{v})$ with $\underline{u}(2j) = \underline{v}(2j)$ 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 $\underline{w}$ of length $|\underline{w}| = 2d$ with $(val_1(\underline{w}) \times val_2(\underline{w})) \cap F \neq \emptyset$. This is because the runs $(\underline{u},\underline{v})$ of $\mathcal{A}$ of length $|\underline{u}| = |\underline{v}| = 2d$ 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 $\underline{w} \in V$, $|\underline{w}| = 2d$ and $(val_1(\underline{w}) \times val_2(\underline{w})) \cap F \neq \emptyset$. 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 $\underline{u} = \underline{u}(1)\underline{u}(2)\ldots\underline{u}(2d) \in V'$, $\underline{v} = \underline{v}(1)\underline{v}(2)\ldots\underline{v}(2d) \in V'$, and $j \in \{1,\ldots,d\}$, $\underline{u}(2j-1) = \underline{v}(2j-1)$.

For any two nodes $\underline{u} = \underline{u}(1)\underline{u}(2)\ldots\underline{u}(2d) \in V'$ and $\underline{v} = \underline{v}(1)\underline{v}(2)\ldots\underline{v}(2d) \in V'$ with $\underline{u} \neq \underline{v}$, $(\underline{u},\underline{v})$ represents, according to the latter item, a set of runs on $\mathcal{T}(\underline{u}(1),\underline{u}(2),\underline{v}(2)) \cdot \ldots \cdot \mathcal{T}(\underline{u}(2d-1),\underline{u}(2d),\underline{v}(2d))$ with $\underline{u}(2j) \neq \underline{v}(2j)$ for at least one $j \in \{1,\ldots,d\}$, which must be all rejecting. Thus, we have $(val_1(\underline{u}) \times val_2(\underline{v})) \cap F = \emptyset$ for any $\underline{u},\underline{v} \in V'$ with $\underline{u} \neq \underline{v}$. But due to $|V'| > |F|$, there are at least two nodes $\underline{u},\underline{v} \in V'$ with $\underline{u} \neq \underline{v}$ such that $(val_1(\underline{u}) \times val_2(\underline{v})) \cap F \neq \emptyset$, which contradicts the premise. Thus, a product automaton $\mathcal{A}$ over $\widetilde{\Sigma}_{\widehat{N}}$ with $L^-(\mathcal{A}) = \mathcal{T}$ cannot exist. This concludes the proof of Claim 8.46. □

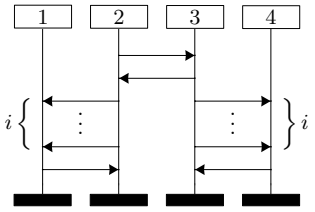


Fig. 8.15. MSC $\mathcal{M}(i)$

Let $N \geq 1$ and set $\widehat{N} = N^2$. For $i \in \{1,\ldots,\widehat{N}+1\}$, consider the MSC $\mathcal{M}(i)$ over $\{1,2,3,4\}$ as illustrated in Fig. 8.15. We claim that the set $L_{\widehat{N}+1} = \{\mathcal{M}(i) \mid i \in \{1,\ldots,\widehat{N}+1\}\}^*$ is contained in $\mathcal{L}((N+1)\text{-}\forall\text{CFM}_\ell) \setminus \mathcal{L}(N\text{-CM})$. Note first that $L_{\widehat{N}+1}$ is $\forall(\widehat{N}+1)$-bounded. The $\forall$-bounded locally accepting $(N+1)$-CFM recognizing $L_{\widehat{N}+1}$ simply sends from agent 2 to agent 3 a message with content $n_1 \in \{1,\ldots,N+1\}$ and then receives a message from agent 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-CM. Suppose there is an N-CM $\mathcal{A} = ((\mathcal{A}_i)_{i\in\{1,2,3,4\}}, \mathcal{D}, \overline{s}^{in}, F)$, $\mathcal{A}_i = (S_i, \Delta_i)$, with $L(\mathcal{A}) = L_{\widehat{N}+1}$. Without loss of generality, we assume that $\mathcal{D} = \{1,\ldots,N\}$. In the following, let $h : \{1,\ldots,N\}^2 \to \{1,\ldots,\widehat{N}\}$ be a bijective mapping and let, as in the proof of Claim 8.46, $A_{\widehat{N}+1} = \{a_1,\ldots,a_{\widehat{N}+1}\}$ and $B_{\widehat{N}+1} = \{b_1,\ldots,b_{\widehat{N}+1}\}$ be alphabets and the distributed alphabet $\widetilde{\Sigma}_{\widehat{N}}$ 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 $s_4 \in S_4$,
- if
$$(s_2, (2!1, i_1)\ldots(2!1, i_k)(2?1, i_{k+1}), s_2') \in \Delta_2 \text{ and}$$
$$(s_1, (1?2, i_1)\ldots(1?2, i_k)(1!2, i_{k+1}), s_1') \in \Delta_1$$
for some $k \in \{1,\ldots,\widehat{N}+1\}$, $i_1,\ldots,i_k,i_{k+1} \in \{1,\ldots,N\}$, $s_2, s_2' \in S_2$, and $s_1, s_1' \in S_1$, then $(s_1, s_2) \overset{a_k}{\Longrightarrow_1} (s_1', s_2')$,
- if
$$(s_3, (3!4, i_1)\ldots(3!4, i_k)(3?4, i_{k+1}), s_3') \in \Delta_3 \text{ and}$$
$$(s_4, (4?3, i_1)\ldots(4?3, i_k)(4!3, i_{k+1}), s_4') \in \Delta_4$$
for some $k \in \{1,\ldots,\widehat{N}+1\}$, $i_1,\ldots,i_k,i_{k+1} \in \{1,\ldots,N\}$, $s_3, s_3' \in S_3$, and $s_4, s_4' \in S_4$, then $(s_3, s_4) \overset{b_k}{\Longrightarrow_2} (s_3', s_4')$.

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

- $S_1' = S_1 \times S_2$ and $S_2' = S_3 \times S_4$,
- $\overline{s}^{in'} = ((\overline{s}^{in}[1], \overline{s}^{in}[2]), (\overline{s}^{in}[3], \overline{s}^{in}[4]))$, 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 $L^-(\mathcal{A}') = \mathcal{T}$ with $\mathcal{T}$ taken from Claim 8.46. But as Claim 8.46 states, such a product automaton $\mathcal{A}'$ does not exist, resulting in a contradiction. □

Lemma 8.47.

$$\mathcal{L}(1\text{-}\forall\text{CFM}) \setminus \mathcal{L}(\text{CM}_\ell) \neq \emptyset$$

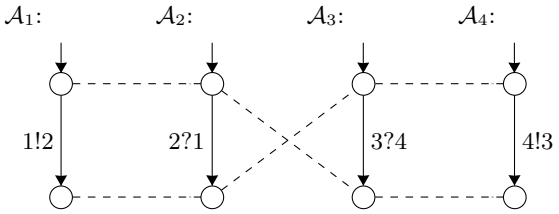


Fig. 8.16. A 1-CFM recognizing L

Proof. Let L consist of the MSCs G and I given by Fig. 7.6 on page 99. Then, L is contained in $\mathcal{L}(1\text{-}\forall\text{CFM}) \setminus \mathcal{L}(\text{CM}_\ell)$. As agents 1 and 2 do not receive any message from 3 and 4 and vice versa, any locally accepting CM accepting both G and I will also accept $G \cdot I$ and $\mathbf{1}_{\mathbb{MSC}}$. In contrast, the $\forall$-bounded 1-CFM from Fig. 8.16 recognizing L has some global knowledge employing global final states. □

We did not pay special attention to the relation between (weak) EMSO-definable product languages and the classes of languages defined by (locally accepting) N-CFMs for $N \geq 2$, which is indicated by the light-gray line in Fig. 8.13 on page 145. 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 8.42.

8.12 Bibliographic Notes

Communicating finite-state machines were first studied in [19]. However, there is no agreement on which automata model is the right one to make an MSC language *realizable*: while, for example, [37] is based on a local acceptance condition and allows us to send synchronization messages, [4, 73] forbid those extra messages. In [45], though a priori unbounded channels are allowed, a channel-bounded model, equipped with a global acceptance mode, suffices to implement regular MSC languages. In the end, the choice of a suitable automata model depends on a thorough analysis of the system environment.

Recall that product languages and related automata models were introduced for traces in [88]. Alur et al. define similar notions for MSCs [4], which then lead to product MSC languages as introduced in [16]. An alternative approach has recently been provided by Adsul et al., who consider the *causal closure* of MSC languages instead of projections onto agents [2]. The local projection onto an agent i is hereby replaced with the downwards closure of the maximal event of i. It turns out that then the question if a regular MSC language is a weak product language is decidable, whereas it is undecidable

when using local projections [4]. The decidability result is based on the observation that the causal closure of a regular MSC language is still regular, which is witnessed by a corresponding $\forall$-bounded locally accepting CFM [2].

The notion of regular MSC languages and the need for synchronization messages were established in [45] and resumed in [56] to extend it in a way to cope with infinite MSCs. Graph acceptors over MSCs were considered in [17], which also studies the expressiveness of CFMs in terms of EMSO logic.

9

Beyond Implementability

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

9.1 EMSO vs. MSO in the Bounded Setting

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

Theorem 9.1. *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 CFM restricted to $\forall B$-bounded MSCs is regular. More precisely, for any CFM $\mathcal{A}$ and any $B \geq 1$, $L(\mathcal{A}) \cap \mathbb{MSC}_{\forall B}$ is a regular MSC language. With Theorem 8.33 and results from [45] and [56], 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 $\varphi(Y_1, \ldots, Y_n) \in \text{MSO}$, $n \geq 1$, there is a CFM $\mathcal{A}$ (adapted to structures from $\langle \mathbb{MSC}, \{0,1\}^n \rangle$) such that $L(\mathcal{A}) = L_{\mathbb{MSC}_{\forall B}}(\varphi)$. So let $\varphi(Y_1, \ldots, Y_n) \in \text{MSO}$, which, according to Lemma 3.5, can be assumed to be of the form

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

or, equivalently,

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

for some $k \geq 1$. We proceed by induction on k. For $k = 1$, φ is an EMSO-formula, which has an equivalent CFM 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 $\Gamma \times \{0,1\}^n$ recognizing $Lin(L(\mathcal{A}) \cap \langle \mathbb{MSC}_{\forall B}, \{0,1\}^n \rangle)$, which is a witness for the fact that $L(\mathcal{A}) \cap \langle \mathbb{MSC}_{\forall B}, \{0,1\}^n \rangle$ is a regular MSC language. According to [74], there is $\mathcal{A}' \in$ det-$\forall$CFM with $L(\mathcal{A}') = L(\mathcal{A}) \cap \langle \mathbb{MSC}_{\forall B}, \{0,1\}^n \rangle$. (Though we did not explicitly define what determinism means for extended CMs, it is obvious how to adjust the definition accordingly so that, then, the above-mentioned result by Mukund et al. also holds in the extended setting.) Induction now alternately involves complementation and projection steps. A complementation step first requires the construction of the CFM $\overline{\mathcal{A}'}$ from $\mathcal{A}'$ with $L(\overline{\mathcal{A}'}) = \langle \mathbb{MSC}, \{0,1\}^n \rangle \setminus L(\mathcal{A}')$, which, though taking into account that we deal with extended MSCs, can be found along the usual lines: we first provide a complete deterministic CFM, whose set of global final states is then complemented. Projection is even easier, as communication actions just need to be projected onto the remaining components. □

As sets of $\forall B$-bounded MSCs can be seen as sets of Mazurkiewicz traces [56] and, in the setting of Mazurkiewicz traces, MSO logic is expressively equivalent to asynchronous automata (cf. Theorem 6.22), Theorem 8.33 can be understood as an extension of Zielonka's theorem.

Proposition 9.2 ([56]). *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 9.2, Theorem 9.1 can be sharpened as follows:

Theorem 9.3. *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. $L \in \mathcal{CFM}$.

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

Theorem 9.4 ([35]). *Theorem 9.3 holds for $\exists$-bounded MSC languages verbatim.*

The proof by Genest et al. makes use of ideas from [56]: existentially bounded MSC languages are also seen as trace languages, which allows us to apply asynchronous mappings [27].

9.2 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 CFM.

Matz and Thomas proved that the monadic quantifier-alternation hierarchy over grids is infinite [66, 93] (cf. Theorem 3.29). We show how grids can be encoded into MSCs and then rewrite the result by Matz and Thomas in terms of MSCs, adapting their proof to our setting.

Theorem 9.5. *The monadic quantifier-alternation hierarchy over* $\mathbb{MSC}$ *is infinite.*

Proof. A grid $\mathcal{G}(n,m)$ can be folded to an MSC $\mathcal{M}(n,m)$ over $\{1,2\}$ as exemplified for $\mathcal{G}(3,5)$ in Fig. 9.1. A similar encoding was used in [92] 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 agent 1 or 2. Formally, $\mathcal{M}(n,m)$ is given by its projections as follows:

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

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

A grid language $\mathcal{G}$ defines the MSC language $L(\mathcal{G}) := \{\mathcal{M}(n,m) \mid \mathcal{G}(n,m) \in \mathcal{G}\}$. 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 [93], which summarizes the result of [66]. So, for $k \in \mathbb{N}$, let 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 $f_{k+1}(n) = f_k(n) \cdot 2^{f_k(n)}$.

Claim 9.6. For each $k \in \mathbb{N}$, the MSC language $L(f_k)$ is $(\Sigma_{2k+3})_{\mathbb{MSC}}$-definable.

Proof of Claim 9.6. We will show that, for any $k \geq 1$, if a grid language $\mathcal{G}$ is $\Sigma_k(-,\{1,2\})_{\mathbb{GR}}$-definable, then $L(\mathcal{G})$ is $\Sigma_k(\Gamma(\{1,2\}))_{\mathbb{MSC}(\{1,2\})}$-definable. The claim then follows from the fact that any grid language $\mathcal{G}(f_k)$ is $(\Sigma_{2k+3})_{\mathbb{GR}}$-definable [93].

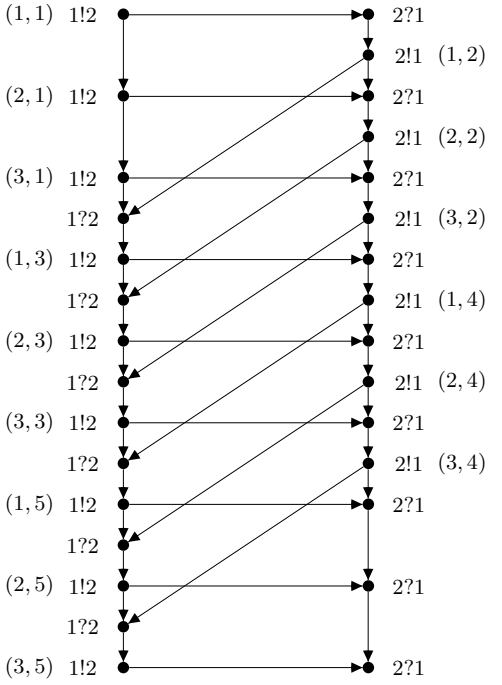


Fig. 9.1. Folding the $(3, 5)$-grid

So let $k \in \mathbb{N}_{\geq 1}$. Figure 9.2 on the facing page shows the CFM $\mathcal{A}_{\mathcal{GF}}$, 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 agents 1 and 2. So, according to Theorem 8.33, let $\varphi_{\mathcal{GF}} = \exists \overline{X} \psi_{\mathcal{GF}}(\overline{X})$ be an EMSO-sentence (over MSCs over $\{1, 2\}$) 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 distinct 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

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

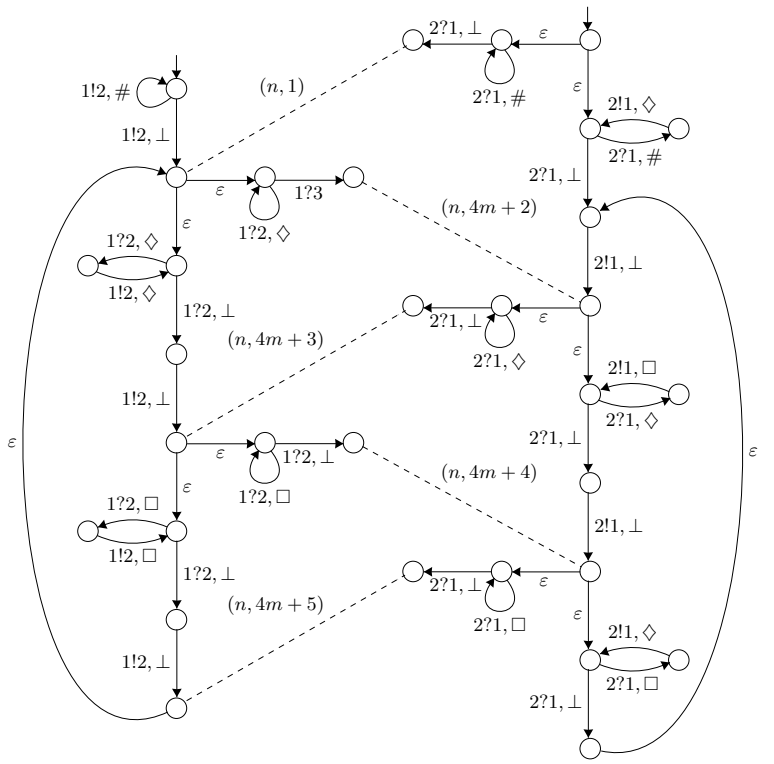


Fig. 9.2. A CFM recognizing $\mathcal{GF}$

Here, the first-order formula bottom(Z) with free variable Z makes sure that Z is reserved to those send vertices that correspond to the end of a column (for simplicity, Z may contain some receive events, too), which are highlighted in Fig. 9.3 for $\mathcal{M}(3,5)$. 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:

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

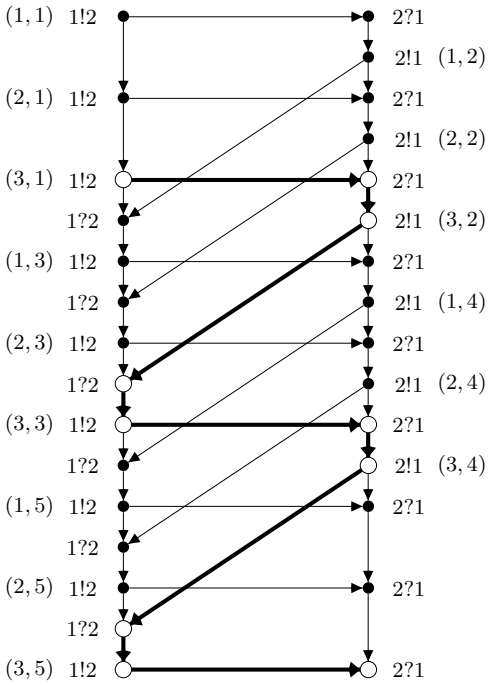


Fig. 9.3. Events of a grid folding that correspond to the end of a column

- $\|S_2(x,y)\|_Z =$
  $\lambda(x) = 1!2 \wedge \lambda(y) = 2!1 \wedge \exists z(x \lhd_c z \wedge z \lhd_2 y)$
  $\vee\ \lambda(x) = 2!1 \wedge \lambda(y) = 1!2 \wedge \exists z(x \lhd_c z \wedge z \lhd_1 y)$
- $\|x \in X\|_Z = x \in X$
- $\|\neg\varphi_1\|_Z = \neg\|\varphi_1\|_Z$
- $\|\varphi_1 \vee \varphi_2\|_Z = \|\varphi_1\|_Z \vee \|\varphi_2\|_Z$
- $\|\varphi_1 \wedge \varphi_2\|_Z = \|\varphi_1\|_Z \wedge \|\varphi_2\|_Z$
- $\|\varphi_1 \to \varphi_2\|_Z = \|\varphi_1\|_Z \to \|\varphi_2\|_Z$
- $\|\varphi_1 \leftrightarrow \varphi_2\|_Z = \|\varphi_1\|_Z \leftrightarrow \|\varphi_2\|_Z$
- $\|\exists x\varphi_1\|_Z = \exists x(\lambda(x) \in \{1!2, 2!1\} \wedge \|\varphi_1\|_Z)$
- $\|\forall x\varphi_1\|_Z = \forall x(\lambda(x) \in \{1!2, 2!1\} \to \|\varphi_1\|_Z)$

Note that the above derivation makes sure that only elements that correspond to grid nodes are assigned to $\overline{Y_1}, \ldots, \overline{Y_k}$. □

Claim 9.7. Let $f : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ be a function. If $L(f)$ is $(\Sigma_k)_{\mathbb{MSC}}$-definable for some $k \geq 1$, then $f(n)$ is in $s_k(\mathcal{O}(n))$.

Proof of Claim 9.7. Let $k \geq 1$ and, in the following, let the events of an MSC $(V, \lhd, \lambda)$ be labeled with elements from $\Gamma \times \{0,1\}^l$ for some $l \in \mathbb{N}_{\geq 1}$, i.e.,

$\lambda : E \to \Gamma \times \{0,1\}^l$. But note that the type of an event still depends on the type of its communication action only. Furthermore, let $\varphi(Y_1, \ldots, Y_l)$ 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 automaton $\mathcal{A}_n$ over $(\Gamma \times \{0,1\}^l)^n$ with $s_{k-1}(c^n)$ states (for some constant c) that reads grid-folding MSCs column by column and is equivalent to $\varphi(Y_1, \ldots, Y_l)$ with respect to 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 $\mathcal{M}(3,5)$ as shown in Fig. 9.1 and 9.3, $\mathcal{A}_3$ 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 $\varphi(Y_1, \ldots, Y_l)$ 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 $\varphi(Y_1, \ldots, Y_l)$ is of the form

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

or, equivalently,

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

We proceed by induction on k. For $k = 1$, $\varphi(Y_1, \ldots, Y_l)$ is an EMSO-formula. According to Theorem 3.24, its MSC language (consisting of MSCs with extended labelings) coincides with the MSC language of some graph acceptor. The transformation from graph acceptors to CFMs can be easily adapted to handle the extended labeling. Thus, $\varphi(Y_1, \ldots, Y_l)$ defines a language that is recognized by some CFM $\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \mathcal{D}, \overline{s}^{in}, F)$. The automaton $\mathcal{A}_n$ can now be obtained from $\mathcal{A}$ using a part of its global transition relation $\Longrightarrow_{\mathcal{A}} \subseteq \mathit{Conf}_{\mathcal{A}} \times (\Gamma \times \{0,1\}^l) \times \mathcal{D} \times \mathit{Conf}_{\mathcal{A}}$. 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 CFM $\mathcal{A}$ and, thus, on $\varphi(Y_1, \ldots, Y_l)$. 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 9.7. □

As $f_{k+1}(n)$ is not in $s_k(\mathcal{O}(n))$, it follows from Claims 9.6 and 9.7 that the hierarchy of classes of $(\Sigma_k)_{\mathbb{MSC}}$-definable MSC languages is infinite. □

Corollary 9.8.

$$\mathcal{CFM} \;=\; \mathcal{EMSO}_{\mathbb{MSC}} \;\subsetneq\; \mathcal{MSO}_{\mathbb{MSC}}$$

As, for any $f : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ and $(V, \lhd, \lambda) \in L(f)$, $\lhd = \lessdot$, which is first-order definable in terms of $\leq$, we obtain the following (cf. Theorem 8.41):

Corollary 9.9. *$\mathcal{MSO}[\leq]_{\mathbb{MSC}}$ and $\mathcal{EMSO}_{\mathbb{MSC}}$ are incomparable with respect to inclusion.*

Since $\mathcal{CFM} = \mathcal{EMSO}_{\mathbb{MSC}}$, it follows that the complement $\mathbb{MSC} \setminus L$ of an MSC language $L \in \mathcal{CFM}$ is not necessarily contained in $\mathcal{CFM}$. Thus, we get the answer to a question, which had been raised by Kuske [56].

Theorem 9.10. *$\mathcal{CFM}$ is not closed under complementation.*

9.3 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 CFMs suffice to achieve the full expressive power of general CFMs. We have learned 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 and $(\widetilde{\Sigma}, C)$-dags in general. Let us first have a look at the bounded setting.

Theorem 9.11 ([56, 74]).

$$\mathcal{L}(\text{det-}\forall\text{CFM}) \;=\; \mathcal{L}(\forall\text{CFM})$$

The algorithm by Mukund et al. to construct from a nondeterministic CFM a deterministic counterpart is based on a technique called *time stamping*, whereas Kuske's construction relies on asynchronous mappings for traces. Unfortunately, the preceding result cannot be transferred to the unbounded setting.

Theorem 9.12.

$$\mathcal{L}(\text{det-CFM}) \;\subsetneqq\; \mathcal{L}(\text{CFM})$$

Proof. According to the algorithm from page 121, we can assume a deterministic CFM $\mathcal{A} = ((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, F)$ to be complete in the sense that, for any MSC $\mathcal{M}$, it allows exactly one run on $\mathcal{M}$. If we set $\overline{\mathcal{A}}$ to be the deterministic CFM $((\mathcal{A}_i)_{i\in Ag}, \mathcal{D}, \overline{s}^{in}, S_{\mathcal{A}} \setminus F)$, then $L(\overline{\mathcal{A}}) = \mathbb{MSC} \setminus L(\mathcal{A})$. Thus, $\mathcal{L}$(det-CFM) is closed under complementation. However, as Theorem 9.10 states, $\mathcal{L}$(CFM) is not closed under complementation, which implies the theorem. □

Theorems 9.10 and 9.12 show that both EMSO logic and CFMs in their unrestricted form are unlikely to have some nice algorithmic properties that would attract practical interest.

Exercise 9.13. Show Theorems 5.50, 5.51, and 5.52.

9.4 CFMs vs. Recognizability

In the field of MSCs, recognizability was first studied by Morin in [73] and turned out to be closely related to implementability.

Proposition 9.14 ([73]). *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 [35], every finitely generated MSC language that is also $\mathrm{MSO}[\leq]_{\mathbb{MSC}}$-definable admits an implementation in terms of a CFM.

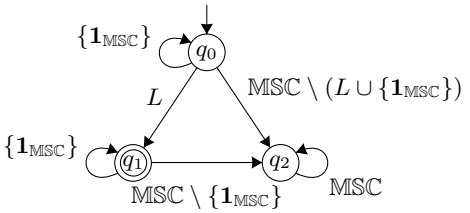


Fig. 9.4. An $\mathbb{MSC}$-automaton

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

Theorem 9.15.

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

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 Fig. 9.4. Conversely, suppose $\mathcal{A} = ((\mathcal{A}_i)_{i \in Ag}, \mathcal{D}, \overline{s}^{in}, F)$ to be a CFM. For a global state $\overline{s} \in S_{\mathcal{A}}$ of $\mathcal{A}$ and an MSC $\mathcal{M}$, we denote by $\widehat{\delta}(\overline{s}, \mathcal{M})$ the set of global states $\overline{s}' \in S_{\mathcal{A}}$ such that $\mathcal{A}$ admits some run on $\mathcal{M}$ that starts in $\overline{s}$ and ends in $\overline{s}'$ (in particular, $(\overline{s}, \chi_\varepsilon) \rightsquigarrow_{\mathcal{A}}^{*} (\overline{s}', \chi_\varepsilon)$). 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 $\mathcal{M}$, $\delta(S, \mathcal{M}) = \bigcup_{\overline{s} \in S} \widehat{\delta}(\overline{s}, \mathcal{M})$, 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 2.7, 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.

9.5 CFMs vs. Rational MSC Languages

A major goal regarding high-level constructs has been to identify specifications that allow implementations in terms of CFMs, preferably automatically and efficiently. This is difficult in general, as an implementation is controlled locally rather than globally, whereas high-level descriptions such as HcMSCs allow us to declare global states that control multiple agents 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 into how an implementation is structured. The former question of deriving an implementation from a given specification is raised in [37, 42], for example, while the inverse problem was tackled in [21, 40, 77]. However, we first show that completeness is not achievable if we consider classes of HcMSCs.

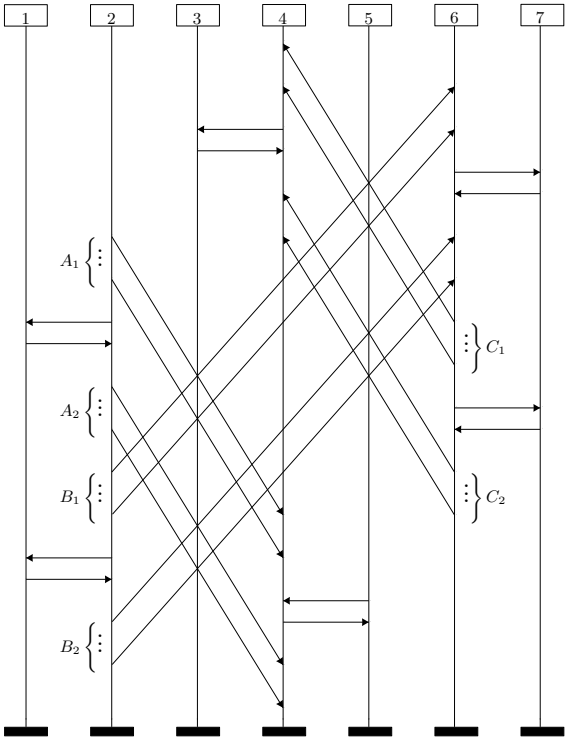


Fig. 9.5. An MSC language that is implementable but not the MSC language of some HcMSC

Proposition 9.16. *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 Fig. 9.5 where the A_i, B_i, C_i are supposed to be natural numbers 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. 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 $\mathcal{M}_A$ that contains a complete message transfer from 2 to 4. By the same argument, there have to appear prime MSCs $\mathcal{M}_B$ and $\mathcal{M}_C$ 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 partial MSCs $\mathcal{M}_A$, $\mathcal{M}_B$, and $\mathcal{M}_C$ we identified cannot contribute to an MSC as depicted in Fig. 9.5, which is a contradiction. □

Generalizing a result by Genest et al., who showed that any safe gc-HcMSC can be defined in terms of $\mathrm{MSO}_{\mathbb{MSC}}$ [35], we show that any gc-HcMSC describes an $\mathrm{MSO}_{\mathbb{MSC}}$-definable MSC language.

Theorem 9.17.

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

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 $\Pi = \{a_1, \ldots, a_m\}$ to be the (finite) set of those prime partial MSCs, which gives rise to the trace monoid $\mathbb{TR}^-(\widetilde{\Sigma}_\Pi)$. Let α be a copy of $\mathcal{H}$, which will then be seen as a rational expression of $\mathbb{TR}^-(\widetilde{\Sigma}_\Pi)$. According to Chap. 6, we can assume the existence of an $\mathrm{MSO}(\Pi)$-sentence φ that defines $L(\alpha)$ relative to $\mathbb{TR}^-(\widetilde{\Sigma}_\Pi)$, i.e., $L(\alpha) = L_{\mathbb{TR}^-(\widetilde{\Sigma}_\Pi)}(\varphi)$. We now construct a formula $\Phi \in \mathrm{MSO}(\Gamma)$ such that $L_{\mathbb{MSC}}(\Phi) = \mathfrak{L}(\mathcal{H})$ as follows:

We extend Π towards $\Pi' = \{a_1, \ldots, a_m, a'_1, \ldots, a'_m\}$, which contains for each member a_k of Π a distinct copy a'_k. Moreover, we define a new distributed alphabet $\widetilde{\Sigma}'_\Pi$ to be $(\Sigma'_i)_{i \in Ag}$ where, for any $i \in Ag$, $\Sigma'_i = \{a \in \Pi' \mid i \in Ag(a)\}$. 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_k with its copy a'_k is $\mathrm{MSO}(\Pi')_{\mathbb{TR}^-(\widetilde{\Sigma}'_\Pi)}$-definable, say by a sentence α'. In particular, for any *trace* $(V, \lhd, \lambda) \in L(\alpha')$ and $u, u' \in V$, $u \lhd u'$ implies $\lambda(u) \neq \lambda(u')$. Though a'_k is a distinct copy of a_k, it will stand for the same prime partial MSC as a_k. This will prove useful

when we now transform α' into the formula $\Phi \in \mathrm{MSO}(\Gamma)$ that defines $\mathcal{L}(\mathcal{H})$ relative to $\mathbb{MSC}$. It is basically of the form

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

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_k} and $X_{a'_k}$ 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 $a \in \Pi'$ to be the prime partial MSC G from Fig. 7.6 on page 99 (recall that a_k and a'_k 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 with respect to $\lhd$ 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 with respect to $\lhd$ 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 $\mathrm{MSO}(\Gamma)$-formula $\psi(X)$. Let us define an order $\prec$ on $\mathcal{X}$ according to $X_k \prec X_l$ if there are $x \in X_k$ and $y \in X_l$ both located on some agent i such that $x \lhd_i y$. 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 $X_k, X_l \in \mathcal{X}$ with $X_k \subseteq X_a$ and $X_l \subseteq X_b$ for some a and b such that both $X_k \prec X_l$ and there is no set $X_r \in \mathcal{X}$ with $X_k \prec X_r \prec^+ X_l$. 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 $\lhd_i$ (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 Π), 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 (even existentially) unbounded behavior. In [35], it is already shown that any safe gc-HcMSC gives rise to an implementable MSC language, supplementing Theorem 9.3.

Theorem 9.18 ([35]). *For any ∃-bounded MSC language L,*

$$L \in \text{gc-}\mathcal{HMSC} \text{ implies } L \in \mathcal{CFM}$$

Actually, [35] even shows that an ∃-bounded MSC language is implementable iff it is the MSC language of some safe HcMSC in which iteration occurs only over partial MSCs with connected communication graph.

9.6 Summary

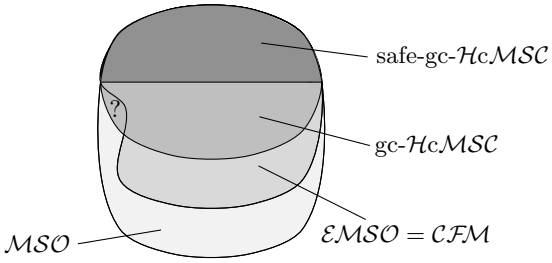


Fig. 9.6. An overview

Some results of this and the previous chapter are illustrated in Fig. 9.6. Moreover, Table 9.1 summarizes the most important results for CFMs. Recall that they are expressively equivalent to EMSO logic over MSCs. In particular, any EMSO sentence admits some implementation in terms of a CFM. The proof was based on results from Chap. 5. Moreover, the class of MSO-definable MSC languages was proved to be strictly larger than the class of implementable ones, concluding that CFMs cannot be complemented in general. We can also infer that the deterministic model of a CFM is strictly weaker than the nondeterministic one.

In the following, we list several open problems of interest that deserve further investigation:

Table 9.1. Closure and expressiveness properties of communicating finite-state machines

$\cup$	$\cap$	$\overline{\cdot}$	det	EMSO	MSO	Empt.
✓	✓	✗	$\supsetneq$	$=$	$\subsetneq$	✗

- It remains open whether there is an EMSO[$\leq$]-sentence (or even FO[$\leq$]-sentence) whose language is not in $\mathcal{CFM}$. 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 $\mathcal{CFM}$ is.
- A further step might be to extend our results to infinite MSCs (and infinite $(\widetilde{\Sigma}, C)$-dags in general). In [56], 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 CMs is primarily a hierarchy of *weakly* realizable MSC languages [4], as their implementation is not necessarily free from deadlocks. It would be desirable to study deadlock-free CMs 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 CFM.
- In [35], 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 [26, 58, 89, 95]. In the framework of MSCs, similar studies are just starting [69, 70]. Of particular interest will be to find a canonical linear-time temporal logic, for example in the sense of Kamp's theorem [26, 51].

9.7 Bibliographic Notes

The study of the gap between MSO over graphs and its existential fragment was initiated in [66] and then extended in [65]. Moreover, [93] provides an accessible overview of that topic. Recognizable MSC languages, CMs, and MSO logic over MSCs have been compared first in [73]. The precise coincidence of universally bounded CFMs and (E)MSO logic has been established in [45]; that between existentially bounded CFMs and (E)MSO logic has been shown in [35].

References

1. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS 2003), Ottawa, Canada*, pages 160–170. IEEE Computer Society Press, 1993.
2. B. Adsul, M. Mukund, K. Narayan Kumar, and Vasumathi Narayanan. Causal closure for MSC languages. In *Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2005)*, volume 3821 of *Lecture Notes in Computer Science*, pages 335–347, Hyderabad, India, 2005. Springer.
3. 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*, pages 797–808. Springer, 2001.
4. R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. *IEEE Trans. Software Eng.*, 29(7):623–633, 2003.
5. R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. *Theoretical Computer Science*, 331(1):97–114, 2005.
6. 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*, pages 114–129. Springer, 1999.
7. J. Araújo. Formalizing sequence diagrams. In *Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.
8. A. Ayari, D. Basin, and A. Podelski. LISA: A specification language based on WS2S. In *Proceedings of Computer Science Logic, 11th International Workshop, CSL 1997, Annual Conference of the EACSL, Aarhus, Denmark*, volume 1414 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 1997.
9. 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.
10. N. Baudru and R. Morin. Safe implementability of regular message sequence chart specifications. In *Proceedings of the ACIS 4th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2003), Lübeck, Germany*, volume 2380 of *Lecture Notes in Computer Science*, pages 210–217. Springer, 2003.

11. 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*, pages 99–114. Springer, 2004.
12. 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.
13. Specification of the Bluetooth System (version 1.1), 2001. http://www.bluetooth.com.
14. B. Bollig. On the expressiveness of asynchronous cellular automata. In *Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT 2005)*, volume 3623 of *Lecture Notes in Computer Science*, pages 528–539, Lübeck, Germany, 2005. Springer.
15. B. Bollig, C. Kern, M. Schlütter, and V. Stolz. MSCan – A tool for analyzing MSC specifications. In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006), Vienna, Austria*, volume 3920 of *LNCS*, pages 455–458. Springer, 2006. The tool web page is located at `http://www-i2.informatik.rwth-aachen.de/MSCan/`.
16. B. Bollig and M. Leucker. A hierarchy of implementable MSC languages. In *Proceedings of the 25th IFIP WG6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, volume 3731 of *Lecture Notes in Computer Science*, pages 53–67, Taipei, Taiwan, ROC, 2005. Springer.
17. B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science*, to appear, 2006. A preliminary version appeared in *Proceedings of CONCUR 2004*, volume 3170 of Lecture Notes in Computer Science, pages 146–160. Springer, 2004.
18. 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.
19. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.
20. J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.
21. T. Chatain, L. Hélouët, and C. Jard. From automata networks to HMSCs: A reverse model engineering perspective. In *Proceedings of the 25th IFIP WG6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, volume 3731 of *Lecture Notes in Computer Science*, pages 489–502, Taipei, Taiwan, ROC, 2005. Springer.
22. 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.
23. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
24. R. Cori, Y. Métivier, and W. Zielonka. Asynchronous mappings and asynchronous cellular automata. *Information and Computation*, 106:159–202, 1993.

25. W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19:1:45–80, 2001.
26. 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.
27. V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
28. Volker Diekert. *Combinatorics on Traces*, volume 454 of *Lecture Notes in Computer Science.* Springer, 1990.
29. M. Droste, P. Gastin, and D. Kuske. Asynchronous cellular automata for pomsets. *Theoretical Computer Science*, 247(1-2):1–38, 2000.
30. H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* second edition. Springer, 1994.
31. Werner Ebinger. Logical definability of trace languages. In Diekert and Rozenberg [27], chapter 10, pages 382–390.
32. C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.
33. J. Fanchon and R. Morin. Regular sets of pomsets with autoconcurrency. In *13th International Conference on Concurrency Theory (CONCUR 2002), Brno, Czech Republic*, volume 2421 of *Lecture Notes in Computer Science*, pages 402–417. Springer, 2002.
34. B. Genest. Compositional message sequence charts (CMSCs) are better to implement than MSCs. In *Proceedings of the 11th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005), Edinburgh, UK*, volume 3340 of *LNCS*, pages 429–444. Springer, 2005.
35. 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*, pages 30–48. Springer, 2004.
36. 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*, pages 195–210. Lecture Notes in Computer Science, 2004.
37. 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*, pages 657–668. Springer, 2002.
38. 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.
39. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, logics, and infinite games*, volume 2500 of *Lecture Notes in Computer Science.* Springer, 2002.
40. E. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. *International Journal on Software Tools and Technology Transfer (STTT)*, 5(1):78–89, 2003.

41. W. 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.
42. D. Harel and H. Kugler. Synthesizing state-based object systems from LSCs specifications. *Foundations of Computer Science*, 13:1:5–51, 2002.
43. L. Hélouët and P. Le Magait. Decomposition of message sequence charts. In *Proceedings of SAM 2000, 2nd Workshop on SDL and MSC, Col de Porte, Grenoble, France*, pages 47–60. VERIMAG, IRISA, SDL Forum, 2000.
44. 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.
45. J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, 2005. The paper subsumes results from [46, 47, 74].
46. 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*, pages 675–686. Springer, 2000.
47. 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 on Mathematical Foundations of Computer Science (MFCS 2000), Bratislava, Slovakia*, volume 1893 of *Lecture Notes in Computer Science*, pages 405–414. Springer, 2000.
48. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
49. ITU-TS Recommendation Z.120anb: Formal Semantics of Message Sequence Charts, 1998.
50. ITU-TS Recommendation Z.120: Message Sequence Chart 1999 (MSC99), 1999.
51. H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.
52. 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.
53. 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.
54. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, December 1983.
55. D. Kuske. Emptiness is decidable for asynchronous cellular machines. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000), University Park, PA, USA*, volume 1877 of *Lecture Notes in Computer Science*, pages 536–551. Springer, 2000.
56. D. Kuske. Regular sets of infinite message sequence charts. *Information and Computation*, 187:80–109, 2003.
57. P. B. Ladkin and S. Leue. Interpreting message flow graphs. *Formal Aspects of Computing*, 7(5):473–509, 1995.

58. M. Leucker. *Logics for Mazurkiewicz traces.* PhD thesis, Lehrstuhl für Informatik II, RWTH Aachen, 2002.
59. 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*, pages 253–264. Springer, 2002.
60. L. Libkin. *Elements of Finite Model Theory.* Springer, Berlin, 2004.
61. M. Lohrey. Realizability of high-level message sequence charts: closing the gaps. *Theoretical Computer Science*, 309(1-3):529–554, 2003.
62. M. Lohrey and A. Muscholl. Bounded MSC Communication. *Information and Computation*, 189(2):160–181, 2004.
63. 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*, pages 809–820. Springer, 2001.
64. 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*, pages 256–267. Springer, 2001.
65. O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. *Information and Computation*, 179(2):356–383, 2002.
66. 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*, pages 236–244. IEEE Computer Society Press, 1997.
67. S. Mauw. The formalization of message sequence charts. *Computer Networks and ISDN Systems*, 28(12):1643–1657, 1996.
68. S. Mauw and M. A. Reniers. High-level message sequence charts. In *Proceedings of the Eighth SDL Forum (SDL'97)*, pages 291–306, 1997.
69. 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*, pages 487–498. Springer, 2000.
70. B. Meenakshi and R. Ramanujam. Reasoning about layered message passing systems. *Computer Languages, Systems, and Structures*, 30(3-4):529–554, 2004.
71. R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.
72. R. Morin. On regular message sequence chart languages and relationships to Mazurkiewicz trace theory. In *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2001), Genova, Italy*, volume 2030 of *Lecture Notes in Computer Science*, pages 332–346. Springer, 2001.
73. 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*, pages 523–534. Springer, 2002.

74. 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*, pages 521–535. Springer, 2000.
75. 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 293–307. Springer, 2003.
76. 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*, pages 81–91. Springer, 1999.
77. 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*, pages 720–731. Springer, 2001.
78. 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*, pages 226–242. Springer, 1998.
79. E. Ochmański. Recognizable Trace Languages. In Diekert and Rozenberg [27], chapter 6, pages 167–204.
80. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
81. D. Perry. *VHDL*. McGraw-Hill, New York, 1991.
82. Giovanni Pighizzini. Asynchronous automata versus asynchronous cellular automata. *Theoretical Computer Science*, 132(2):179–207, 1994.
83. Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS 1977), Providence, Rhode Island*, pages 46–57. IEEE Computer Society Press, 1977.
84. A. Potthoff, S. Seibert, and W. Thomas. Nondeterminism versus determinism of finite automata over directed acyclic graphs. *Bulletin of the Belgian Mathematical Society*, 1, 1994.
85. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier, 2001.
86. Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, September 2002.
87. B. Sengupta and R. Cleaveland. Triggered message sequence charts. In *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 167–176. ACM Press, 2002.
88. P. S. Thiagarajan. PTL over product state spaces. Technical Report TCS-95-4, Chennai Mathematical Institute, 1995.
89. 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*, pages 438–452. Springer, 1995.

90. W. Thomas. On logical definability of trace languages. In *Proceedings of a workshop of the ESPRIT Basic Research Action No 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS), Kochel am See, Germany (1989)*, Report TUM-I9002, Technical University of Munich, pages 172–182, 1990.
91. 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.
92. 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.
93. 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*, pages 20–38. Springer, 1997.
94. W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 389–455. Springer, Berlin, 1997.
95. I. Walukiewicz. Difficult configurations – on the complexity of LTrL. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP 1998), Aalborg, Denmark*, volume 1443 of *Lecture Notes in Computer Science*, pages 140–151, 1998.
96. P. Weil. Algebraic recognizability of languages. In *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS 2004), Prague, Czech Republic*, volume 3153 of *Lecture Notes in Computer Science*, pages 149–175. Springer, 2004.
97. W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.

Symbols and Notation

Chapter 2

$\mathcal{R}_1 \circ \mathcal{R}_2$	relation product	11
$\lessdot$	covering relation	12
$\mathcal{RAT}_{\mathbb{M}}$	rational subsets of $\mathbb{M}$	12
$\mathcal{REC}_{\mathbb{M}}$	recognizable subsets of $\mathbb{M}$	14
$\square$	blank symbol of a Turing machine	14
$Conf_{\mathcal{M}}$	configurations of a Turing machine	14
$\vdash_{\mathcal{M}}$	behavior of a Turing machine	14

Chapter 3

$\lhd$	edge relation	17
$\lvert\mathcal{G}\rvert$	cardinality of $\mathcal{G}$	17
$\mathbb{DG}$	set of graphs	18
$\mathcal{K}[B]$	graphs from $\mathcal{K}$ with degree bounded by B	18
$\langle\mathcal{K}, Q\rangle$	Q-extension of $\mathcal{K}$	18
MSO	monadic second-order formulas	18
FO	first-order formulas	20
EMSO	existential monadic second-order formulas	20
Σ_k	Σ_k-formulas	20
$\mathcal{MSO}_{\mathcal{K}}$	MSO-definable languages	21
$\mathcal{FO}_{\mathcal{K}}$	FO-definable languages	21
$\mathcal{EMSO}_{\mathcal{K}}$	EMSO-definable languages	21
$\mathcal{L}_{\mathcal{K}}(\Sigma_k)$	Σ_k-definable languages	21
$d_{\mathcal{G}}(u', u)$	distance from u' to u	23
R-Sph$(\mathcal{G}, u)$	R-sphere of $\mathcal{G}$ around u	23
R-Sph$(\mathcal{K})$	R-spheres that arise from $\mathcal{K}$	24
$rank(\varphi)$	rank of formula φ	24

$\mathcal{G}_1 \equiv_k \mathcal{G}_2$	FO-k-equivalence	24
$\mathcal{G}_1 \leftrightarrows_{R,t} \mathcal{G}_2$	threshold equivalence	25
$Cond(\mathcal{O}, t)$	occurrence conditions over $\mathcal{O}$ and t	25
$\mathcal{LTT}_\mathcal{K}$	locally threshold testable languages	26
$h(\mathcal{G})$	projection of $\mathcal{G}$	27
$L_\mathcal{K}(\mathcal{B})$	language of $\mathcal{B}$ relative to $\mathcal{K}$	28
$\mathcal{GA}_\mathcal{K}$	languages of GAs	28
k-$\mathcal{GA}_\mathcal{K}$	languages of GAs with k-spheres	28
$\mathcal{GA}^-_\mathcal{K}$	languages of GAs without occurrence constraints	29
$\mathbb{DAG}$	set of dags	30
$\mathbb{DAG}_H$	set of Hasse dags	30
$\mathcal{G} \Downarrow u$	downwards closure	30
$\mathcal{G} \downarrow u$	strict downwards closure	30
$\mathbb{P}$	pictures	31
$\mathbb{GR}$	grids	33

Chapter 4

$\mathbb{W}$	words	35
$first(\underline{w})$	first element of $\underline{w}$	35
$last(\underline{w})$	last element of $\underline{w}$	35
ε	empty word	35
$\mathcal{FA}$	languages of finite automata	37
det-$\mathcal{FA}$	languages of deterministic finite automata	37
$Lin(\mathcal{G})$	linearizations of $\mathcal{G}$	39

Chapter 5

Ag	agents	44
$\widetilde{\Sigma}$	distributed alphabet	44
$loc(a)$	agents involved in a	44
$I_{\widetilde{\Sigma}}$	independence relation	44
$D_{\widetilde{\Sigma}}$	dependence relation	44
Ch	channels	44
$\mathbb{DAG}_{lo}(\widetilde{\Sigma}, C)$	lo-dags	45
$\mathbb{DAG}(\widetilde{\Sigma}, C)$	$(\widetilde{\Sigma}, C)$-dags	45
$\mathbb{DAG}_{\rightrightarrows}(\widetilde{\Sigma}, C)$	fifo-dags	45
$\mathbb{DAG}(\widetilde{\Sigma})$	$(\widetilde{\Sigma}, -)$-dags	45
Read(u)	read domain of u	47

$\mathrm{Write}(u)$	write domain of u	47
$\mathcal{D} \restriction i$	projection of $\mathcal{D}$ onto i	47
$\lambda_{\mid V'}$	restriction of λ to V'	47
$\mathbb{TR}^+$	M^+-traces	48
$\mathbb{TR}^-$	M^--traces	48
Γ	communication actions over Ag	48
$\widetilde{\Gamma}$	distributed alphabet over Ag	48
$\mathbb{MSC}$	MSCs	49
$\mathbb{LMSC}$	lossy MSCs	49
$\Longrightarrow_{(\widetilde{\Sigma},C)}$	operational behavior of $(\widetilde{\Sigma}, C)$-dags	50
$\mathit{Trans}_{(\widetilde{\Sigma},C)}(Q)$	transitions over $(\widetilde{\Sigma}, C)$ and Q	50
$\mathcal{D}(t)$	graph of transition t	50
$\overline{q} \longrightarrow (a, q) \in \Delta$	$(\overline{q}, a, q) \in \Delta$	51
$\mathit{final}^{\overline{q}}_{\mathcal{D}}$	global final state	52
$\mathit{source}^{\overline{q}}_{\mathcal{D}}(u)$	source nodes of u	52
$\mathcal{ACAT}_{\mathcal{K}}$	languages of ACATs	53
$\mathcal{ACA}_{\mathcal{K}}$	languages of ACAs	53

Chapter 6

$\mathbb{TR}^+$	M^+-traces	77
$\mathbb{TR}^-$	M^--traces	77
$\mathbf{1}_{\mathbb{TR}^\alpha}$	unit trace	79
c-$\mathcal{RAT}_{\mathbb{TR}^\alpha}$	c-rational subsets of $\mathbb{TR}^\alpha$	79
$\mathcal{R}_{\mathbb{TR}^\alpha}$	regular trace languages	79
$L \vdash_{\widetilde{\Sigma}} \mathcal{T}$	trace inference	79
$\mathcal{P}^0_{\mathbb{TR}^\alpha}$	weak product trace languages	80
$\mathcal{P}_{\mathbb{TR}^\alpha}$	product trace languages	80
$\mathcal{RP}^0_{\mathbb{TR}^\alpha}$	weak regular product trace languages	80
$\mathcal{RP}_{\mathbb{TR}^\alpha}$	regular product trace languages	80
$\overline{s} \xrightarrow{a} \overline{s}'$	$(\overline{s}, \overline{s}') \in \Delta_a$	81
$\mathcal{AA}^\alpha$	languages of asynchronous automata	83
det-$\mathcal{AA}^\alpha$	languages of det. asynchronous automata	83
$\Longrightarrow_{\mathcal{A}}$	global transition relation of $\mathcal{A}$	83
$\mathcal{PA}^\alpha$	languages of product automata	88

Chapter 7

Γ	communication actions over Ag	91
$\widetilde{\Gamma}$	distributed alphabet over Ag	92
V_{um}	unmatched nodes	92

$Ag(u)$	agent executing u	92
$Ag(\mathcal{M})$	agents of $\mathcal{M}$	92
$\mathbb{PMSC}$	partial MSCs	93
$\mathbb{LMSC}$	lossy MSCs	93
$\mathbb{MSC}$	MSCs	93
$\mathcal{M} \preceq \mathcal{M}'$	$\mathcal{M}$ represents $\mathcal{M}'$	93
$\mathrm{CG}(\mathcal{M})$	communication graph of $\mathcal{M}$	95
$\mathbb{MSC}_{\forall B}$	universally B-bounded MSCs	96
$\mathbb{MSC}_{\exists B}$	existentially B-bounded MSCs	96
$\mathcal{L}(\mathcal{H})$	MSC language of HcMSC $\mathcal{H}$	98
$\mathcal{G}raph(\mathcal{H})$	graph of HcMSC $\mathcal{H}$	98
$\mathcal{HMSC}$	MSC languages of HcMSCs	100
gc-$\mathcal{HMSC}$	MSC languages of gc-HcMSCs	100
safe-gc-$\mathcal{HMSC}$	MSC languages of safe gc-HcMSCs	100
left-closed-gc-$\mathcal{HMSC}$	MSC languages of left-closed gc-HcMSCs	100
$\mathcal{HMSC}$	MSC languages of HMSCs	100
gc-$\mathcal{HMSC}$	MSC languages of gc-HMSCs	100
lc-$\mathcal{HMSC}$	MSC languages of lc-HcMSCs,	102
$\mathbb{MSC}_{\not\searrow}(Ag, \Lambda)$	(potentially) non-fifo MSCs over (Ag, Λ)	104
$\mathbb{MSC}(Ag, \Lambda)$	fifo MSCs over (Ag, Λ)	104
$\Box(\mathcal{M}, L)$	universal LSC	105
$\Diamond(\mathcal{M})$	existential LSC	105
$\mathcal{R}_{\mathbb{MSC}}$	regular MSC languages	107
$L \vdash_{Ag} \mathcal{M}$	MSC inference	109
$\mathcal{P}^0_{\mathbb{MSC}}$	weak product MSC languages	109
$\mathcal{P}_{\mathbb{MSC}}$	product MSC languages	109
$\mathcal{RP}^0_{\mathbb{MSC}}$	weak regular product MSC languages	110
$\mathcal{RP}_{\mathbb{MSC}}$	regular product MSC languages	110
$\mathcal{EP}^0_{\mathbb{MSC}}$	weak EMSO-def. product MSC languages	110
$\mathcal{EP}_{\mathbb{MSC}}$	EMSO-def. product MSC languages	110

Chapter 8

CM	CMs	118
CFM	CFMs	118
N-CM	N-CMs	118
CM_ℓ	locally-accepting CMs	118
det-CM	deterministic CMs	118
$\mathcal{L}(\mathrm{CM})$	languages of CMs	118
$\mathcal{CFM}$	languages of CFMs	118
$Conf_{\mathcal{A}}$	configurations of $\mathcal{A}$	119
$\Longrightarrow_{\mathcal{A}}$	global transition relation of $\mathcal{A}$	119
$\rightsquigarrow_{\mathcal{A}}$	global transition relation of $\mathcal{A}$	120

$\forall$CM	$\forall$-bounded CMs	122
$\exists$CM	$\exists$-bounded CMs	122
$\forall\forall$CM	strongly $\forall$-bounded CMs	122
safe-CM	safe CMs	122
$\overset{\ell}{\rightsquigarrow}_{\mathcal{A}}$	lossy steps of $\mathcal{A}$	128
$\rightsquigarrow^{lcs}_{\mathcal{A}}$	global transition relation of LCS $\mathcal{A}$	128
$\mathcal{CFM}_{\not\searrow}(Ag, \Lambda)$	languages of (potentially) non-fifo CFMs	130
$\mathcal{CFM}(Ag, \Lambda)$	languages of fifo CFMs over (Ag, Λ)	130

Index