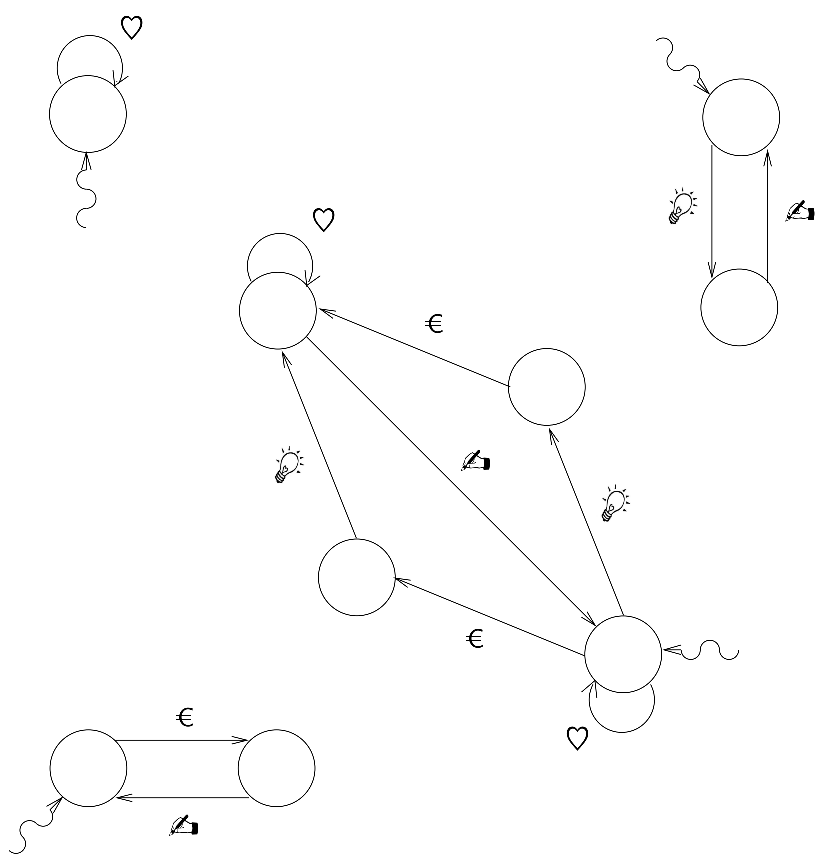
Author: Beek, Maurice H. ter
Title: Team automata : a formal approach to the modeling of collaboration between system components
Issue Date: 2003-12-10

Team Automata

A Formal Approach to the Modeling of Collaboration Between System Components

Maurice H. ter Beek



Team Automata

A Formal Approach to the Modeling of Collaboration Between System Components

Proefschrift
ter verkrijging van
de graad van Doctor
aan de Universiteit Leiden,
op gezag van de Rector Magnificus Dr. D.D. Breimer,
hoogleraar in de faculteit der Wiskunde en
Natuurwetenschappen en die der Geneeskunde,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 10 december 2003
te klokke 15.15 uur
door

Maurice Henri ter Beek
geboren te 's-Gravenhage in 1972

Promotiecommissie

Promotor:	Prof.dr. G. Rozenberg
Copromotor:	Dr. H.C.M. Kleijn
Referent:	Prof.dr. C.A. Ellis (University of Colorado at Boulder, U.S.A.)
Overige leden:	Prof.dr. Th. Bäck
	Prof.dr. G. van Dijk
	Prof.dr. J.N. Kok
	Prof.dr. M. Koutny (University of Newcastle upon Tyne, U.K.)

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

voor John en Lia

Acknowledgments

I would never have become the person I am without the continuous and unconditional love and support of my parents... pa en ma, *bedankt*!

Part of the research for this thesis was conducted outside of Leiden, most notably in Pisa and Budapest. In Pisa I was initially hosted by Fabrizio Luccio at the Department of Computer Science of the University of Pisa and later by Stefania Gnesi at the Institute of Science and Information Technology of the National Research Council. In Budapest I was hosted by Erzsébet Csuhaj-Varjú at the Computer and Automation Research Institute of the Hungarian Academy of Sciences. I am very grateful for the enduring hospitality and friendship provided by my colleagues at these institutes.

Notwithstanding my frequent trips abroad, the bulk of the research for this thesis was of course carried out in Leiden at LIACS. During all the years I spent there as a member of the Theoretical Computer Science group, my trips back to Leiden have always remained something to look forward to. For this I thank my former group members and other colleagues at LIACS.

I must admit that during the last few years the progress of my thesis has been (too) frequently the subject of conversation between me and my friends. In fact, I suspect some of them to be more relieved than me now that it is finished! But seriously, the genuine interest of my friends has always stimulated me enormously and for this I thank them all very much. I consider myself lucky to have too many friends to list them here one by one. Let me make one exception and thank Vincent for a friendship that goes beyond brotherhood.

Finally, the person that has supported me most during all the years I have worked on this thesis — and that continues to do so in my daily life — is *mijn lief*. Nadia, *ti amo*!

Table of Contents

1. Introduction

This thesis studies formal aspects of *team automata*, a mathematical framework introduced in [Ell97] to model components of *groupware systems* and their interconnections. In particular, this thesis focuses on the flexibility team automata offer when modeling collaboration between system components.

We begin this Introduction by providing some background. Subsequently we introduce the model in an informal way, after which we discuss its main features in the context of several related models. Finally, we finish this Introduction with an overview of the contents of this thesis.

Background

A set of interacting, interrelated, or interdependent components forming a complex whole is what we mean by the frequently used, but seldom defined notion of a *system*. The human body and computers are thus examples of a system. A system is *distributed* if it consists of separate components but nevertheless appears to its users as a single coherent system. It does not have a single locus of control, but its components collaborate by way of interactions. The internet is one of the best known distributed systems.

A system is *reactive* if, in order for it to function, it has a continuous need to interact with its environment. Its functioning thus depends on the functioning of its environment. This contrasts with a system that is *transformational*, in which case its functioning (output) is merely a function of its input. Examples of reactive systems include computer operating systems and coffee vending machines, whereas a compiler is an example of a transformational system.

Computer Supported Cooperative Work

As the presence of computer-based systems in daily-life work situations continues to increase, the understanding of how people work together and ways in which computer technology can assist, has become more and more important.

This has resulted in the emergence of *Computer Supported Cooperative Work* (CSCW for short) as an inherently multi-disciplinary field of research (see, e.g., [Gru94]). By the nature of the field, part of the computer technology consists of multi-user software and hardware, called *groupware*.

Groupware systems are systems intended to support groups of people working together in collaborative projects. Such systems are often distributed and reactive, and conceived as consisting of components cooperating in a coordinated way. This leads to complex interactive behavior and, consequently, coordination policies and their effect on behavior are key issues within CSCW. At a conceptual level CSCW needs a precise, consistent, and unambiguous terminology, while at a lower, architectural level CSCW has been searching for a rigorous mathematical framework to specify and verify groupware systems.

Formal Methods

Mathematical techniques tailored for the specification and verification of systems are known as *formal methods* (see, e.g., [CW96]). This field of research cuts across many areas of computer science and comes with an impressive body of literature. A brief comparison of the main features of team automata with some of the best-known formalisms in this field follows later on in this Introduction, while a more detailed comparison with two such formalisms can be found in Chapter 7.

The model of *Input/Output automata* (I/O automata for short) was introduced in [Tut87] for the specification and verification of distributed reactive systems (see also, e.g., [LT89] and [Lyn96]). I/O automata served as the theoretical source of inspiration for the introduction of team automata in [Ell97] through the distinction of the model's actions into input, output, and internal actions. We come back to this shortly. A conceptual source of inspiration for team automata was [Smi94], which conjectures that well-structured groups (called teams) outperform individuals in certain tasks, but at the same time calls for models capturing concepts of group behavior.

Team automata were introduced explicitly for the specification and verification of groupware systems. They were shown to be promising at both the conceptual and the architectural level of groupware systems. In this thesis we elaborate on this. Our goal is furthermore to demonstrate that the usefulness of team automata is not limited to clarifying and capturing precisely notions related to collaboration between components of groupware sytems, but extends to other kinds of (reactive) systems.

The Model

We now provide an overview of the team automata framework. We begin with a brief sketch of the overall structure of team automata and subsequently we introduce them in more detail. Analogous to the setup of this thesis, we follow an incremental presentation of team automata.

A team automaton is composed of *component automata*, which are a special type of *automata*. The crux of composing a team automaton is to define the way in which those originally independent component automata interact. Their interactions are formulated in terms of *synchronizations* of shared actions, a method for modeling collaboration among system components well known from the literature.

Automata

Automata or *labeled transition systems* are a well-known model underlying formal specifications of systems. An automaton consists of a set of states, a set of actions, a set of labeled transitions between states, and a set of initial states. Labels represent actions and a transition's label indicates the action causing the transition from one state to another.

Assume that we have an automaton modeling a coffee vending machine. Then a possible event is a user inserting a coin, which when it occurs leads to a state change of the automaton. The user forms a part of the environment of the coffee vending machine. A coffee vending machine is thus an example of a reactive system, with the insertion of coins by a user as interactions with its environment.

Next assume that also the user is modeled by an automaton, with the insertion of a coin as one of its actions. Then we have two automata, both equipped with an action modeling the insertion of a coin. When composing these two automata into one system, inserting a coin into the coffee vending machine appears as a single synchronized action. In the composed system the occurrences of an action from the automaton modeling the user and the same action from the automaton modeling the coffee vending machine are identified, i.e. simultaneously executed by the two system components. The transitions of a thus composed automaton will be synchronized occurrences of transitions of its constituting automata that have the same action label.

Synchronized Automata

A *synchronized automaton* over a set of automata is an automaton, determined by the way in which its constituting automata cooperate by means

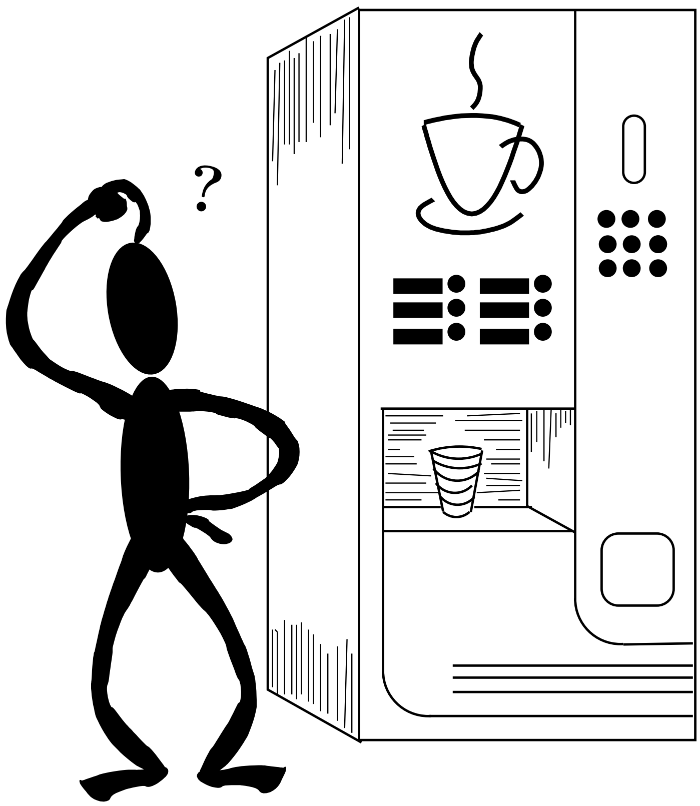


Fig. 1.1. A user in front of a coffee vending machine.

of synchronized transitions. Its (initial) states are combinations — a cartesian product — of (initial) states of its constituting automata. Its actions are the actions of its constituting automata. Its transitions, finally, are *synchronizations* of labeled transitions of its constituting automata modeling the simultaneous execution of the same single action by several (one or more) automata. The label of a transition is the action being simultaneously executed. When the synchronized automaton changes state by executing an action, all automata which participate simultaneously change state by executing that action, while all others remain idle.

An automaton does not necessarily participate in every synchronization of an action it shares. Hence there is no such thing as the unique synchronized automaton over a set of automata. Rather, a whole range of synchronized automata, distinguishable only by their transition relation, can be constructed from a given set of automata. It is this freedom to choose a transition relation that sets the team automata framework apart from most other models. Another distinguishing feature of this framework is the fact that the transitions of a synchronized automaton are labeled with one single action. We come back to this shortly.

From the way a synchronized automaton is constructed it is clear that it is itself an automaton again. Consequently, it can serve as a constituting automaton of a higher-level synchronized automaton, thus allowing hierarchical designs.

Within a synchronized automaton, three natural types of actions can be distinguished, based on the way they appear in synchronizations. Actions that are never executed simultaneously by more than one constituting automaton are *free*. Actions that are always executed as synchronizations in which all automata participate that have this action in their alphabet are called *action-indispensable*. *State-indispensable* actions, finally, require the participation of only those automata that are ready (in a suitable state) to execute that action.

Team Automata

A component automaton is an automaton in which *input*, *output*, and *internal actions* are distinguished. Input actions are not under the automaton's control, but instead are triggered by the environment including other component automata. Output and internal actions are under its control, but only the output actions are observable by other automata. Input and output actions together constitute the *external actions* and they form the interface between the automaton and its environment, whereas the internal actions are not available for interactions. This is formally achieved by requiring that the internal actions of each component automaton involved are unique to that automaton, which naturally prohibits synchronizations of internal actions with other automata.

A team automaton over a set of component automata is defined in a way similar to the definition of synchronized automata. As before, its (initial) states are cartesian products of (initial) states of its constituting component automata. Its actions are the actions of its constituting component automata, now distributed over input, output, and internal actions. All internal (output) actions of the component automata remain internal (output) actions of the team automaton. The remaining actions are those input actions of the component automata that do not occur as an output action of any of the component automata, and they become the input actions of the team automaton. Its labeled transitions, finally, are — as before — synchronizations of labeled transitions of its constituting component automata.

Like in the case of synchronized automata, we do not require all constituting component automata sharing an action to participate in every synchronization of that action. Synchronizations of internal actions never involve more than one component automaton because every internal action uniquely belongs to one particular component automaton. Moreover, independently of the states of the other component automata, an internal action can always be executed as before the composition. Like in the case of synchronized automata, there is no unique team automaton. Rather a whole range of team automata, distinguishable only by their transition relation, can be constructed.

The reason given in [Ell97] for equipping team automata — like I/O automata — with a distinction of actions into input, output, and internal actions, is the explicit desire to model different types of synchronization. This is achieved by taking the different role (input, output, or internal) that actions can have in different component automata into account. External actions may be input to some component automata and output to other component automata. In *peer-to-peer* synchronizations, actions have the same role in each of the component automata involved. In such synchronizations, all component automata are on equal footing with respect to the action being synchronized. This differs from *master-slave* synchronizations, in which input actions ("slaves") are driven by output actions ("masters"), i.e. the slaves have to follow the masters.

Team automata form a very broad and generic framework. Component automata can cooperate in many possible ways through synchronizations of shared actions. The freedom of choosing the transition relation of a team automaton moreover offers the flexibility to distinguish even the smallest nuances in the meaning of one's design. Leaving the set of transitions of a team automaton as a modeling choice thereby becomes one of the most important features of team automata. One of the topics of this thesis is a systematic study of the role of free, action-indispensable, and state-indispensable actions — and to a lesser degree peer-to-peer and master-slave synchronizations — in the modeling of collaboration between system components.

Team Automata Versus Other Models

Team automata are not an isolated model but have several features which bear a close resemblance to characteristics of other models from the literature. We now discuss three such features in general terms.

First, the set of actions of a team automaton consists of input, output, and internal actions, thus allowing the classification of a broad range of often complex synchronizations in team autamata (cf. Sections 4.4 and 5.3). This distinction of input, output, and internal actions originates from two independently developed models: I/O automata (see, e.g., [Tut87], [LT89], and [Lyn96]) and *I/O systems* (see, e.g., [Jon87] and [Jon94]). Since the semantics of an I/O system — given in terms of automata — is essentially an I/O automaton, we will speak only of I/O automata in the sequel. Team automata are, in fact, an extension of I/O automata (cf. Section 7.1).

I/O automata are not the only model in the literature in which a distinction of actions is used. The same distinction can be found in the I/O automata-based *reactive transition systems* (see, e.g., [CC02] and [CCP02]) as well as in *interacting state machines* (see, e.g., [Ohe03] and [OL02]), which were introduced specifically for modeling reactive systems. A further example is the *Calculus of Communicating Systems* (CCS for short), an algebraic specification language introduced by Milner (see, e.g., [Mil80] and [Mil89]). In CCS, the internal or *silent action* τ is a distinguished element of the set of actions. It denotes the "perfect" action of a *handshake communication*, i.e. the synchronization of two *complementary (input and output) actions*.

Secondly, the transitions of a team automaton are synchronizations of transitions with the same label. The simultaneous execution of actions from a team automaton's constituting component automata is thus limited to common actions. We call such types of synchronization *uniform* in order to distinguish them from *pluriform synchronizations* in which distinct actions can be executed simultaneously.

Also this feature of allowing solely uniform synchronizations originates from the I/O automaton model. It is by far not the only model in the literature prohibiting pluriform synchronizations. Other examples include the *mixed product* over a set of automata introduced in [Dub86] and the *product automaton* introduced in [TH98]. A further example is the theory of *path expressions*, which was introduced in [CH74], consequently encompassed in the *COncurrent SYstems* (COSY for short) notation in [LTS79], and given a *vector firing sequence* semantics in [Shi79], which considers *vector actions* rather than ordinary actions (see also [JL92]). An entry of such a vector action is not empty if and only if the respective component participates.

There are also examples of automata-based models that do allow pluriform synchronizations, such as the *free product* and the *synchronous product* over a set of automata. Both were defined in [Arn94] as the culmination of a framework of process models proposed by Nivat and Arnold in a number of papers and course notes such as, e.g., [Niv79], [AN82], and [Arn82]. Another example is the framework of *Vector Controlled Concurrent Systems* (VCCSs for short) introduced in [KKR90] (cf. Sections 7.2.3 and 7.2.4). These systems, introduced as generalizations of the COSY theory, allow pluriform synchronizations of actions of its constituting components and execute vectors of actions rather than ordinary actions. In Section 7.2.1 we will switch to vector actions in order to visualize the (potential) concurrency within team automata actions, but such vector actions will still be uniform synchronizations.

Yet another type of synchronization is the handshake communication in CCS mentioned above. Many algebraic specification languages moreover contain specific parallel composition operators that allow processes to communicate through synchronizations (see, e.g., [BPS01]). Among the best known such examples are the *(Theoretical) Communicating Sequential Processes* ((T)CSP for short) originally introduced by Hoare (see, e.g., [Hoa78], [BHR84], and [Hoa85]).

Thirdly, the transition relation of a team automaton is not uniquely determined by its constituting component automata, which also distinguishes team automata from I/O automata. This freedom of choosing the transition relation of the automaton obtained when composing a set of automata, occurs in the literature as well. An example is the aforementioned synchronous product over a set of automata. Whereas the transition relation of the free product over a set of automata is the set of all possible pluriform synchronizations, that of the synchronous product over that set of automata is the restriction of the free product to the subset of all possible pluriform synchronization vectors defined by a specifically formulated synchronization constraint. This synchronization constraint is formulated in terms of the actions only and does not depend on the current states of the automata.

Most automata-based models, however, use a single and very strict method for choosing the transition relation of an automaton composed over a set of automata, in effect resulting in composite automata that are uniquely defined by their constituents. The choice prevalent in the literature is to include, for all actions, all and only those transitions in which all automata participate that have the action in their alphabet. Since this means that all actions will be action-indispensable, we call this the *ai* principle. Examples of automata-based models with composition based on the *ai* principle include the aforementioned mixed product and product automaton over a set of automata, as well as reactive transition systems, interacting state machines, and I/O automata (cf. Section 7.1). Other examples from the literature — without claiming completeness — include *cooperating (pushdown) automata* (see, e.g., [DH94] and [HH94]) and *timed cooperating automata* (see, e.g., [LMP00]). The *ai* principle furthermore appears in disguise in non-automata-based models like (T)CSP and *statecharts* (see, e.g., [Har87]).

In Section 5.4 we define team automata that are unique with respect to particular types of synchronization. Through the formulation of predicates of synchronization we moreover provide direct constructions for such team automata. Throughout the thesis we will see, though, that of all the resulting uniquely defined team automata, it is precisely the one based on the *ai* principle that possesses the at first sight most appealing characteristics. One of the contributions of this thesis is to put some order in the "chaos" obtained when refraining from the *ai* principle. More precisely, we present an overview of some interesting characteristics that hold for certain types of team automata, among which those based on the peer-to-peer and master-slave types of synchronization. Since these types of synchronization are introduced with a clear practical motivation in mind, it is worthwhile to notice that output peer-to-peer as well as master-slave synchronizations cannot be distinguished in I/O automata (cf. Section 7.1). In fact, in a team automaton constructed according to the *ai* principle, all synchronizations are by definition master-slave.

To the best of our knowledge, no automata-based model other than team automata unites the three features discussed above. I/O automata satisfy the first two features, viz. the distinction of input, output, and internal actions, and the prohibition of pluriform synchronizations. However — as already noted in [Tut87] — the single notion of automaton composition in I/O automata is rather restrictive and may hinder a realistic modeling of certain types of interactions. This is the main motivation given in [Ell97] for introducing team automata as a generalization of I/O automata. Another important reason for generalizing I/O automata is the fact that I/O automata are *input enabling*, i.e. in every state of the automaton every input action of that automaton can be executed. Though convenient when modeling reactive computer systems, this hinders a realistic modeling of interactions that involve humans (cf. Section 7.1). Team automata have thus been introduced with the motivation of creating a single model in which the above three features are united.

Origins of the Thesis

This thesis is a monograph which is partly based on papers that were published in various places. Below we list these papers in the order in which they were written.

In [BEKR03] we elaborated further on the concept of team automata, introduced in [Ell97] for modeling groupware systems, by defining team automata in a mathematically precise way. We showed how the formal setup allows one to distinguish between several types of synchronization and to classify team automata accordingly. Based on the observation that team automata can be used as components in higher-level teams, we showed also how the framework allows for the representation of hierarchical systems.

In [HB00] we sketched how team automata can be employed to model collaboration between teams (of humans) engaged in team-based development of (software) configuration management models.

In [BEKR01b] we demonstrated the model usage and utility for capturing information security and protection structures, and critical coordinations between these structures. On the basis of a spatial access metaphor, various known access control strategies were given a rigorous formal description in terms of synchronizations in team automata.

In [BEKR01a] we presented a survey of [BEKR03] and [BEKR01b], augmented with the introduction of team automata with vectors as actions, and a preliminary comparison of team automata with I/O automata and models based on Petri nets.

In [BK03] we presented an initial investigation of the conditions under which team automata satisfy compositionality, in the sense that their behavior can be described in terms of that of their constituting component automata.

Outline of the Thesis

Although this is a theoretical thesis written for theoretical computer scientists interested in formal models with a clear practical motivation, we hope that it is also accessible for practical computer scientists well motivated to look for formalizations of models that can aid in the early design phase of complex systems. In order to achieve this we have generously accompanied our formal definitions and results by explanations and examples, providing the motivation for and the interpretation of these definitions and results.

After this Introduction, we fix most basic notation and terminology used throughout this thesis in Chapter 2. In Chapters 3 and 4 we introduce automata and synchronized automata, respectively. On top of this foundation we then build our team automata framework in Chapter 5. In Chapter 6 we study the behavior of team automata, while Chapter 7 provides a comparison with other models. Before finishing the thesis with a Discussion, we show some of the fields of application of team automata in Chapter 8. We now provide a more detailed description of each of these chapters and, where appropriate, mention the published papers used in that chapter.

In Chapter 3 we define the automata as used in this thesis and we review some notions from automata theory.

In Chapter 4 we define how to combine a set of automata in order to form a synchronized automaton. We also define how to obtain a subautomaton from a synchronized automaton as a subset of its constituting automata, and we study the relation between synchronized automata and their subautomata in terms of computations. Consequently, we show how to compose synchronized automata in an iterative way. Within synchronized automata we then characterize three basic and very natural ways of synchronizing on shared actions of their constituting automata, which form the basis of the more complex types of synchronization we introduce later. Finally, we define unique synchronized automata being maximal with respect to a given type of synchronization. Through the formulation of predicates of synchronization we moreover provide direct constructions of such synchronized automata. Some of the material in this chapter is based on [BEKR03].

In Chapter 5 we define team automata as compositions of component automata, i.e. from now on we distinguish input, output, and internal actions. To this aim we use the foundation laid in the preceding chapters and build team automata and component automata on top of (synchronized) automata. We then build subteams on top of subautomata, and we study the relation between team automata and their subteams. Also in the case of team automata, we show how to compose them in an iterative way. We then build several complex types of synchronization on top of those introduced in the previous chapter, by using the different roles that an action may have in the various component automata. Similar to synchronized automata, we define unique team automata being maximal with respect to particular types of synchronization. Through the formulation of predicates of synchronization we furthermore provide direct constructions for such team automata. Most of the material in this chapter is based on [BEKR03].

In Chapter 6 we study the computations and behavior of team automata in relation to those of their constituting component automata. Therefore we study (synchronized) shuffles and their properties. We prove that the behavior of certain types of team automata can be described in terms of certain (synchronized) shuffles of the behavior of their constituting component automata. Some of this material is based on [BK03].

In Chapter 7 we provide a comparison of team automata with two other models. The first is I/O automata, of which team automata are an extension. The second is a model based on Petri nets, for which we define team automata with vector actions as an extension of team automata. A small part of this material is based on [BEKR03], but most of it is based on [BEKR01a].

In Chapter 8 we present three examples demonstrating the usefulness of team automata in practical settings. Based on [BEKR03], we first show how to model a specific groupware architecture by team automata. Secondly, based on [HB00], we show how team automata can be employed to model collaboration between teams of developers engaged in the development of models of complex (software) systems. Thirdly, based on [BEKR01b], we show how various known access control strategies can be given a rigorous formal description in terms of synchronizations in team automata.

In the Discussion, finally, we recall the main contributions of this thesis and point out some topics worth further investigation. Furthermore, we indicate how — in theory — team automata can be used for system design and where — in practice — they have actually been used.

It is worth mentioning that at the end of this thesis one can find — in addition to the Bibliography and the Index — a List of Figures and a List of Symbols, which should allow one to quickly find the page on which a figure or symbol first appeared.

2. Preliminaries

In this chapter we fix most basic notation and terminology used throughout this thesis.

Sets

Set inclusion is denoted by $\subseteq$, whereas proper inclusion is denoted by $\subset$. The set difference of sets V and W is denoted by $V \setminus W$. For a finite set V, its cardinality is denoted by $\#V$. The empty set is denoted by $\varnothing$. For convenience, we sometimes denote the set $\{1, 2, \ldots, n\}$ by $[n]$. Then $[0] = \varnothing$. We sometimes identify a singleton set $\{j\}$ with its only element j.

Let $\mathbb{N}$ denote the set of positive integers. Let $\mathcal{I} \subseteq \mathbb{N}$ be a set of indices given by $\mathcal{I} = \{i_1, i_2, \ldots\}$ with $i_j < i_\ell$ if $1 \leq j < \ell$ and let V_i be a set, for each $i \in \mathcal{I}$. Then $\prod_{i \in \mathcal{I}} V_i$ denotes the cartesian product $\{(v_{i_1}, v_{i_2}, \ldots) \mid v_{i_j} \in V_{i_j}, \text{ for all } j \geq 1\}$. The elements of $\prod_{i \in \mathcal{I}} V_i$ are called vectors. If $\mathcal{I}$ is finite and $\#\mathcal{I} = n$, then the vectors in $\prod_{i \in \mathcal{I}} V_i$ are said to be n-dimensional. Throughout this thesis vectors may be written vertically as well as horizontally. If $v_i \in V_i$, for all $i \in \mathcal{I}$, then $\prod_{i \in \mathcal{I}} v_i$ denotes the element $(v_{i_1}, v_{i_2}, \ldots)$ of $\prod_{i \in \mathcal{I}} V_i$. If $\mathcal{I} = \varnothing$, then $\prod_{i \in \mathcal{I}} V_i = \varnothing$. In addition to the prefix notation $\prod_{i \in \mathcal{I}} V_i$ for a cartesian product, we sometimes also use the infix notation $V_{i_1} \times V_{i_2} \times \cdots$.

Let $j \in \mathcal{I}$. Then $\mathrm{proj}_{\mathcal{I},j} : \prod_{i \in \mathcal{I}} V_i \to V_j$ is the projection function defined by $\mathrm{proj}_{\mathcal{I},j}((a_{i_1}, a_{i_2}, \ldots)) = a_j$. We thus observe that if $\mathcal{I} = \{2, 3\}$, then $\mathrm{proj}_{\mathcal{I},2}((a, b)) = a$. Note moreover that whenever $\mathcal{I} = \mathbb{N}$, then $\mathrm{proj}_{\mathcal{I},j}$ is the standard projection. Similarly, for $J \subseteq \mathcal{I}$, $\mathrm{proj}_{\mathcal{I},J} : \prod_{i \in \mathcal{I}} V_i \to \prod_{i \in J} V_i$ is the projection function defined by $\mathrm{proj}_{\mathcal{I},J}(a) = \prod_{j \in J} \mathrm{proj}_{\mathcal{I},j}(a)$. Whenever $\mathcal{I}$ is clear from the context we write proj_j and proj_J rather than $\mathrm{proj}_{\mathcal{I},j}$ and $\mathrm{proj}_{\mathcal{I},J}$. Note that for each $j \in \mathcal{I}$ and $a \in \prod_{i \in \mathcal{I}} V_i$ we have $\mathrm{proj}_{\{j\}}(a) = \prod_{j \in \{j\}} \mathrm{proj}_j(a)$, which we do not identify with $\mathrm{proj}_j(a)$. Formally, we have $\mathrm{proj}_j(\mathrm{proj}_{\{j\}}(a)) = \mathrm{proj}_j(a)$.

The set $\{V_i \mid i \in \mathcal{I}\}$ is said to form a partition (of $\bigcup_{i \in \mathcal{I}} V_i$) if the V_i are pairwise disjoint, nonempty sets.

Functions

All functions considered are total, unless explicitly stated otherwise.

Let $f : A \to A'$ and let $g : B \to B'$ be functions. Then $f \times g : A \times B \to A' \times B'$ is defined as $(f \times g)(a, b) = (f(a), g(b))$. We will use $f^{[2]}$ as shorthand notation for $f \times f$. Thus $f^{[2]}(a, b) = (f(a), f(b))$. This notation should not be confused with iterated function application. In particular, we will use ${\mathrm{proj}_{\mathcal{I},j}}^{[2]}$ as shorthand notation for $\mathrm{proj}_{\mathcal{I},j} \times \mathrm{proj}_{\mathcal{I},j}$ and likewise ${\mathrm{proj}_{\mathcal{I},J}}^{[2]}$ for $\mathrm{proj}_{\mathcal{I},J} \times \mathrm{proj}_{\mathcal{I},J}$. We write ${\mathrm{proj}_j}^{[2]}$ and ${\mathrm{proj}_J}^{[2]}$ rather than ${\mathrm{proj}_{\mathcal{I},j}}^{[2]}$ and ${\mathrm{proj}_{\mathcal{I},J}}^{[2]}$ whenever $\mathcal{I}$ is clear from the context. If $C \subseteq A$, then $f(C) = \{f(a) \mid a \in C\}$. Thus if $D \subseteq A \times A$, then $f^{[2]}(D) = \{(f(d_1), f(d_2)) \mid (d_1, d_2) \in D\}$.

The function f is injective if $f(a_1) \neq f(a_2)$ whenever $a_1 \neq a_2$, f is surjective if for every $a' \in A'$ there exists an $a \in A$ such that $f(a) = a'$, and f is a bijection if f is injective and surjective. The restriction of the function f to a subset C of its domain A is denoted by $f \restriction C$ and is defined as the function $C \to A'$ defined by $(f \restriction C)(c) = f(c)$, for all $c \in C$.

Alphabets, Words, Languages

An alphabet is a set of letters — symbols — which may be used, e.g., to represent actions of systems. We do not impose any a priori constraints on the size of an alphabet. Alphabets may thus be empty and they may be infinite. For the remainder of this chapter we let Σ be an arbitrary but fixed alphabet.

A word (over Σ) is a sequence of symbols (from Σ). A word may be a finite or infinite sequence of symbols, resulting in finite and infinite words, respectively. An infinite word is also referred to as an ω-word. The empty sequence is called the empty word and denoted by λ. As usual we represent nonempty words $a_1, a_2, \ldots$ over Σ as strings $a_1a_2 \cdots$. For a finite word w, we use the notation $|w|$ to denote its length. Thus $|\lambda| = 0$ and if $w = a_1a_2 \cdots a_n$, with $n \geq 1$ and $a_i \in \Sigma$, for all $1 \leq i \leq n$, then $|w| = n$.

Words may also be considered as functions which assign symbols to positions. Thus a finite word $w = a_1a_2 \cdots a_n$, with $n \geq 1$ and $a_i \in \Sigma$ for all $1 \leq i \leq n$, is identified with the function $w : [n] \to \Sigma$ defined by $w(i) = a_i$, for all $1 \leq i \leq n$. Similarly, an infinite word $w = a_1a_2 \cdots$, with $a_i \in \Sigma$ for all $i \geq 1$, defines the function $w : \mathbb{N} \to \Sigma$ by $w(i) = a_i$, for all $i \geq 1$. To the empty word λ we associate the function $\lambda : [0] \to \Sigma$, which has an empty domain.

For a finite word w over Σ and a symbol $a \in \Sigma$, we use $\#_a(w)$ to denote the number of occurrences of a in w. Thus $\#_a(w) = \#\{i \in [|w|] \mid w(i) = a\}$. Note that $\#_a(\lambda) = 0$, for all a. For a (finite or infinite) word w, its alphabet, denoted by alph(w), consists of all symbols that actually occur in w. Thus alph$(w) = \{a \in \Sigma \mid \exists i \in \mathbb{N} : \ w(i) = a\}$. Note that alph$(\lambda) = \varnothing$ and that alph(w) may be an infinite set if Σ is infinite and w is an infinite word.

The set of all finite words over Σ (including λ) is denoted by Σ^*. The set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ consists of all nonempty finite words. By convention $\Sigma \subseteq \Sigma^+$. The set of all infinite words over Σ is denoted by Σ^ω. By Σ^∞ we denote the set of all words over Σ. Thus $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A language (over Σ) is a set of words (over Σ). A language consisting solely of finite words is called finitary. If $L \subseteq \Sigma^\omega$, i.e. all words of L are infinite, then L is called an infinitary language or ω-language. As usual we refer to a collection (set) of languages as a family of languages.

Concatenation

Using the operation of concatenation, two words (over Σ) are combined into one word (over Σ) by gluing them together.

Formally, given $u, v \in \Sigma^\infty$, their concatenation $u \cdot v$ is defined as follows. If $u, v \in \Sigma^*$, then $u \cdot v(i) = u(i)$ for $i \in [|u|]$ and $u \cdot v(|u| + i) = v(i)$ for $i \in [|v|]$. Note that $|u \cdot v| = |u| + |v|$. If $u \in \Sigma^*$ and $v \in \Sigma^\omega$, then $u \cdot v(i) = u(i)$ for $i \in [|u|]$ and $u \cdot v(|u| + i) = v(i)$ for $i \geq 1$. If $u \in \Sigma^\omega$ and $v \in \Sigma^\infty$, then $u \cdot v(i) = u(i)$ for all $i \geq 1$. In the last two cases $u \cdot v \in \Sigma^\omega$. Note that $u \cdot \lambda = \lambda \cdot u = u$, for all $u \in \Sigma^\infty$. Since concatenation is associative this implies that Σ^∞ with concatenation and unit element λ is a monoid. Moreover, since concatenation of two finite words yields a finite word, also Σ^* with concatenation restricted to Σ^* is a monoid with unit element λ.

The concatenation of two languages K and L (over Σ) is the language $K \cdot L$ (over Σ) defined by $K \cdot L = \{u \cdot v \mid u \in K, \ v \in L\}$. Observe that $K \cdot L$ is finitary if and only if both K and L are finitary. Moreover, $K \cdot L = K$ if $L = \{\lambda\}$ or K is infinitary. In the sequel, we will mostly write uv and KL rather than $u \cdot v$ and $K \cdot L$, respectively.

For $u \in \Sigma^\infty$ we set $u^0 = \lambda$ and $u^{n+1} = u^n \cdot u$, for all $n \geq 0$. Note that if $u \in \Sigma^\omega$, then $u^n = u$, for all $n \geq 1$. Similarly, for a language $K \subseteq \Sigma^\infty$ we have $K^0 = \{\lambda\}$ and $K^{n+1} = K^n \cdot K$, for all $n \geq 0$.

Prefixes

A word $u \in \Sigma^*$ is said to be a (finite) prefix of a word $w \in \Sigma^\infty$ if there exists a $v \in \Sigma^\infty$ such that $w = uv$. In that case we write $u \leq w$. If $u \leq w$ and $u \neq w$, then we may use the notation $u < w$. Moreover, if $|u| = n$, for some $n \geq 0$, then u is said to be the prefix of length n of w, denoted by $w[n]$. Note that $w[0] = \lambda$. The set of all prefixes of a word w is denoted by

$\text{pref}(w)$ and it is defined as $\text{pref}(w) = \{u \in \Sigma^* \mid u \leq w\}$. Note that $\text{pref}(w)$ is finite if and only if $w \in \Sigma^*$. Note also that, for a word $x \in \Sigma^\infty$, whenever $\text{pref}(w) = \text{pref}(x)$, then $w = x$.

For a language K, $\text{pref}(K) = \bigcup\{\text{pref}(w) \mid w \in K\}$. Thus $K \subseteq \text{pref}(K)$ whenever K is a finitary language. A language K is prefix closed if and only if $K \supseteq \text{pref}(K)$. A family of languages L is prefix closed if $\text{pref}(K) \in \mathsf{L}$ for all $K \in \mathsf{L}$.

Limits

Both finite and infinite words can be defined as limits of their prefixes. Let $v_1, v_2, \dots \in \Sigma^*$ be an infinite sequence of words such that $v_i \leq v_{i+1}$, for all $i \geq 1$. Then $\lim_{n\to\infty} v_n$ is the unique word $w \in \Sigma^\infty$ defined by $w(i) = v_j(i)$, for all $i, j \in \mathbb{N}$ such that $i \leq |v_j|$. Thus $v_i \leq w$ for all $i \geq 1$ and $w = v_k$ whenever there exists a $k \geq 1$ such that $v_n = v_{n+1}$ for all $n \geq k$. For a word $u \in \Sigma^\infty$ we define $u^\omega = \lim_{n\to\infty} u^n$ if $u \in \Sigma^*$ and $u^\omega = u$ if $u \in \Sigma^\omega$. Note that $\lambda^\omega = \lambda$. For an infinite sequence $u_1, u_2, \ldots \in \Sigma^\infty$ we define the word $u_1 \cdot u_2 \cdot \cdots \in \Sigma^\infty$ by $u_1 \cdot u_2 \cdot \cdots = \lim_{n\to\infty} u_1 \cdot u_2 \cdot \cdots \cdot u_n$ if $u_i \in \Sigma^*$, for all $i \geq 1$, and $u_1 \cdot u_2 \cdot \cdots = u_1 \cdot u_2 \cdot \cdots \cdot u_{n-1} \cdot u_n$ if $u_n \in \Sigma^\omega$, for some $n \geq 1$.

These notations are carried over to languages in the natural way: for $K, K_1, K_2, \ldots \subseteq \Sigma^\infty$, we set $K^\omega = \{u_1 u_2 \cdots \mid u_i \in K, \text{ for all } i \geq 1\}$ and $K_1 \cdot K_2 \cdot \cdots = \{u_1 u_2 \cdots \mid u_i \in K_i, \text{ for all } i \geq 1\}$. Observe that $\Sigma^\omega = \{a_1 a_2 \cdots \mid a_i \in \Sigma, \text{ for all } i \geq 1\}$ is indeed the set consisting of all infinite words over Σ.

Homomorphisms

Let $h : \Sigma \to \Gamma^*$ be a function assigning to each letter of Σ a finite word over the alphabet Γ. The homomorphic extension of h to Σ^*, also denoted by h, is defined in the usual way by $h(\lambda) = \lambda$ and $h(xy) = h(x)h(y)$ for all $x, y \in \Sigma^*$. This homomorphism is further extended to Σ^∞ by setting $h(\lim_{n\to\infty} v_n) = \lim_{n\to\infty} h(v_n)$, for all $v_1, v_2, \ldots \in \Sigma^*$ such that for all $i \geq 1$, $v_i \leq v_{i+1}$. Note that this is well defined, since $v_i \leq v_{i+1}$ implies $h(v_i) \leq h(v_{i+1})$. Note however that if h is erasing, i.e. $h(a) = \lambda$ for some $a \in \Sigma$, then there exists a word $x \in \Sigma^\omega$ such that $h(x) \in \Sigma^*$. For such x we have $h(xy) = h(x)$, for all $y \in \Sigma^\infty$, and consequently $h(xy) = h(x)h(y)$ is no longer guaranteed. In fact, $h(xy) = h(x)h(y)$, for all $x, y \in \Sigma^\infty$, if and only if either h is not erasing or $h(a) = \lambda$, for all $a \in \Sigma$. Thus $h : \Sigma \to \Gamma^*$ cannot always be lifted to a homomorphism on Σ^∞. Still we sometimes abuse terminology and refer to the extension $h : \Sigma^\infty \to \Gamma^\infty$ of h as a homomorphism. If $h(\Sigma) \subseteq \Gamma$, then we refer to h as a coding, and if $h(\Sigma) \subseteq \Gamma \cup \{\lambda\}$, then h is called a weak coding.

The function $\mathrm{pres}_{\Sigma,\Gamma} : \Sigma \to \Gamma^*$, defined by $\mathrm{pres}_{\Sigma,\Gamma}(a) = a$ if $a \in \Gamma$ and $\mathrm{pres}_{\Sigma,\Gamma}(a) = \lambda$ otherwise, preserves the symbols from Γ and erases all other symbols. Whenever Σ is clear from the context, we simply write pres_{Γ} rather than $\mathrm{pres}_{\Sigma,\Gamma}$. Note that $\mathrm{pres}_{\Sigma,\Gamma}$ is a weak coding.

3. Automata

The basic concept underlying team automata is an *automaton*. An automaton captures the idea of a system with *states* (configurations, possibly an infinite number of them), together with *actions* the executions of which lead to (nondeterministic) state changes. In addition some of the states may be designated as *initial states* from which the automaton may start its executions. Also final or accepting states may be distinguished, which can be used to define when an execution of the automaton is considered successful. A particular automaton model is the well-known *finite (state) automaton*. Such an automaton has a finite set of states, with initial states and final states, as well as a finite set of actions. Finite automata are among the most basic models in many branches of computer science.

In this thesis automata are used as structures defining a state space that is traversed by executing actions. They come into play when designing and analyzing complex systems with a potentially infinite number of configurations due to, e.g., unbounded data structures such as counters.

We begin this chapter by defining precisely the type of automata we shall use in the sequel, thus laying the foundation on which we shall build our team automata framework. Subsequently we review some notions from automata theory.

3.1 Automata, Computations, and Behavior

Definition 3.1.1. *An* automaton *is a construct* $\mathcal{A} = (Q, \Sigma, \delta, I)$*, where*

Q *is the set of* states *of* $\mathcal{A}$*, which may be infinite,*
Σ *is the set of* actions *of* $\mathcal{A}$ *such that* $\Sigma \cap Q = \varnothing$*,*
$\delta \subseteq Q \times \Sigma \times Q$ *is the set of* labeled transitions *of* $\mathcal{A}$*, and*
$I \subseteq Q$ *is the set of* initial states *of* $\mathcal{A}$*.* □

In the figures, the states of an automaton are drawn as circles and labeled transitions appear as labeled arcs between states. Wavy arcs are used to indicate the initial states. See, e.g., Figure 3.1.

Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let $a \in \Sigma$. Then the set of *a-transitions* (of $\mathcal{A}$) is denoted by δ_a and is defined as $\delta_a = \{(q, q') \mid (q, a, q') \in \delta\}$. An a-transition $(q, q) \in \delta_a$ is called a *loop* (on a). We refer to $\mathcal{A}$ as the *trivial automaton* if $\mathcal{A} = (\varnothing, \varnothing, \varnothing, \varnothing)$. Instead of labeled transition we often simply say transition. Finally, a transition $(q, q') \in \delta_a$ is called an *outgoing transition* of q and an *incoming transition* of q'.

Executing an action in a certain state leads to a change of state as described by the labeled transitions. The consecutive execution of a sequence of actions from an initial state defines a *computation*.

Definition 3.1.2. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

(1) a finite computation *of $\mathcal{A}$ is a finite sequence $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n$, where $n \geq 0$, $q_i \in Q$ for $0 \leq i \leq n$, and $a_j \in \Sigma$ for $1 \leq j \leq n$ are such that $q_0 \in I$ and $(q_i, a_{i+1}, q_{i+1}) \in \delta$ for all $0 \leq i < n$;*
if $n = 0$ and hence $\alpha = q_0 \in I$, then α is a trivial *computation;*
by $\mathbf{C}_{\mathcal{A}}$ we denote the set of all finite computations *of $\mathcal{A}$,*

(2) an infinite computation *of $\mathcal{A}$ is an infinite sequence $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots$, where $q_i \in Q$ for all $i \geq 0$ and $a_j \in \Sigma$ for all $j \geq 1$ are such that $q_0 \in I$ and $(q_i, a_{i+1}, q_{i+1}) \in \delta$ for all $i \geq 0$;*
by $\mathbf{C}^{\omega}_{\mathcal{A}}$ we denote the set of all infinite computations *of $\mathcal{A}$, and*

(3) the set of all computations *of $\mathcal{A}$ is denoted by $\mathbf{C}^{\infty}_{\mathcal{A}}$ and is defined as $\mathbf{C}^{\infty}_{\mathcal{A}} = \mathbf{C}_{\mathcal{A}} \cup \mathbf{C}^{\omega}_{\mathcal{A}}$.* □

Thus for a given automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$, its finite computations form a finitary language $\mathbf{C}_{\mathcal{A}} \subseteq I(\Sigma Q)^*$ while its infinite computations form an infinitary language $\mathbf{C}^{\omega}_{\mathcal{A}} \subseteq I(\Sigma Q)^{\omega}$. Observe that $\mathbf{C}_{\mathcal{A}} = \varnothing$ if and only if $I = \varnothing$. Moreover, $\mathbf{C}^{\omega}_{\mathcal{A}}$ may be empty, even when $\mathbf{C}_{\mathcal{A}}$ is infinite (cf. Example 3.1.12).

The infinite computations of $\mathcal{A}$ can be expressed in terms of finite computations, viz. as limits of length-increasing sequences of finite computations.

Lemma 3.1.3. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Let $\alpha \in \mathbf{C}^{\infty}_{\mathcal{A}}$. Then*

$\alpha \in \mathbf{C}^{\omega}_{\mathcal{A}}$ if and only if there exist $\alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{\mathcal{A}}$ such that for all $n \geq 1$, $\alpha_n \neq \alpha_{n+1}$ and $\alpha = \lim_{n \to \infty} \alpha_n$.

Proof. (If) Trivial.
(Only if) Obvious from the observation $\mathrm{pref}(\alpha) \cap I(\Sigma Q)^* \subseteq \mathbf{C}_{\mathcal{A}}$. □

Both finite and infinite computations are thus sequences of which every prefix of odd length is a finite computation.

Theorem 3.1.4. *Let $\mathcal{A}$ be an automaton. Then*

$\alpha \in \mathbf{C}^{\infty}_{\mathcal{A}}$ *if and only if for all* $n \geq 1$ *there exist* $\alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{\mathcal{A}}$ *such that* $\alpha = \lim_{n\to\infty} \alpha_n$. □

In fact, the infinite computations of an automaton are determined by its set of finite computations.

Lemma 3.1.5. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two automata. Then*

if $\mathbf{C}_{\mathcal{A}} \subseteq \mathbf{C}_{\mathcal{A}'}$, *then* $\mathbf{C}^{\omega}_{\mathcal{A}} \subseteq \mathbf{C}^{\omega}_{\mathcal{A}'}$.

Proof. Let $\alpha \in \mathbf{C}^{\omega}_{\mathcal{A}}$. Hence by Lemma 3.1.3, $\alpha = \lim_{n\to\infty} \alpha_n$ for computations $\alpha_n \in \mathbf{C}_{\mathcal{A}}$ such that $\alpha_n \leq \alpha_{n+1}$ and $\alpha_n \neq \alpha_{n+1}$, for all $n \geq 1$. Since $\mathbf{C}_{\mathcal{A}} \subseteq \mathbf{C}_{\mathcal{A}'}$, again applying Lemma 3.1.3 (now in the other direction) yields that $\alpha \in \mathbf{C}^{\omega}_{\mathcal{A}'}$. □

Theorem 3.1.6. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two automata. Then*

$\mathbf{C}^{\infty}_{\mathcal{A}} = \mathbf{C}^{\infty}_{\mathcal{A}'}$ *if and only if* $\mathbf{C}_{\mathcal{A}} = \mathbf{C}_{\mathcal{A}'}$. □

Given a computation of an automaton one may choose to focus on certain actions while filtering away other information. In this way, behavioral records are made of computations.

Definition 3.1.7. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ be an alphabet disjoint from Q. Then*

(1) $v \in \Theta^{\infty}$ *is a* Θ-record *of* $\mathcal{A}$ *if* $v = pres_{\Theta}(\alpha)$ *for some* $\alpha \in \mathbf{C}^{\infty}_{\mathcal{A}}$,

(2) the Θ-behavior *of $\mathcal{A}$ is denoted by* $\mathbf{B}^{\Theta,\infty}_{\mathcal{A}}$ *and is defined as* $\mathbf{B}^{\Theta,\infty}_{\mathcal{A}} = pres_{\Theta}(\mathbf{C}^{\infty}_{\mathcal{A}})$,

(3) the finitary Θ-behavior *of $\mathcal{A}$ is denoted by* $\mathbf{B}^{\Theta}_{\mathcal{A}}$ *and is defined as* $\mathbf{B}^{\Theta}_{\mathcal{A}} = \mathbf{B}^{\Theta,\infty}_{\mathcal{A}} \cap \Theta^*$, *and*

(4) the infinitary Θ-behavior *of $\mathcal{A}$ is denoted by* $\mathbf{B}^{\Theta,\omega}_{\mathcal{A}}$ *and is defined as* $\mathbf{B}^{\Theta,\omega}_{\mathcal{A}} = \mathbf{B}^{\Theta,\infty}_{\mathcal{A}} \cap \Theta^{\omega}$. □

If Σ is the full set of actions of automaton $\mathcal{A}$, then a Σ-record is also simply called a *record* and the (finitary or infinitary) Σ-behavior of $\mathcal{A}$ is also referred to as the (finitary or infinitary) behavior of $\mathcal{A}$, respectively.

W_1:

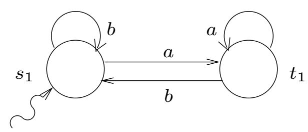


Fig. 3.1. Automaton W_1.

Example 3.1.8. Let $W_1 = (\{s_1, t_1\}, \{a, b\}, \delta_1, \{s_1\})$, where $\delta_1 = \{(s_1, b, s_1), (s_1, a, t_1), (t_1, a, t_1), (t_1, b, s_1)\}$, be an automaton modeling a **w**heel (of a car). It is depicted in Figure 3.1.

The state s_1 indicates that the wheel stands **s**till, while the state t_1 indicates that the wheel **t**urns. The result of **a**ccelerating, modeled by action a, makes the wheel turn. The result of **b**raking, modeled by action b causes the wheel to stand still. Initially the wheel stands still, as indicated by the initial state s_1.

An example of a finite computation of W_1 is $\alpha = s_1 a t_1 b s_1 \in \mathbf{C}_{W_1}$, modeling accelerating and subsequently braking. The record of this computation is $\mathrm{pres}_\Sigma(\alpha) = ab$, which is thus an element of the finitary behavior of W_1: $ab \in \mathbf{B}^{\Sigma}_{W_1}$. An example of an infinite computation of W_1 is $s_1 a t_1 b s_1 b s_1 \cdots \in \mathbf{C}^{\omega}_{W_1}$, which thus leads to an example of an infinitary behavior $ab^\omega \in \mathbf{B}^{\Sigma,\omega}_{W_1}$. □

It is immediate that finite computations define finite records. In fact, all finite Θ-records can be obtained from finite computations. On the other hand, infinite computations may give rise to finite Θ-records even though infinite Θ-records can only be obtained from infinite computations.

Lemma 3.1.9. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ be an alphabet disjoint from Q. Then*

(1) $\mathbf{B}^{\Theta}_{\mathcal{A}} = pres_\Theta(\mathbf{C}_{\mathcal{A}})$ and

(2) $\mathbf{B}^{\Theta,\omega}_{\mathcal{A}} = pres_\Theta(\mathbf{C}^{\omega}_{\mathcal{A}}) \cap \Theta^\omega$.

Proof. (1) ($\supseteq$) Immediate.

($\subseteq$) Let $v \in \Theta^*$ and $\alpha \in \mathbf{C}^{\infty}_{\mathcal{A}}$ be such that $\mathrm{pres}_\Theta(\alpha) = v$. Let $\alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{\mathcal{A}}$ be such that $\alpha = \lim_{n\to\infty} \alpha_n$. Since pres_Θ is a homomorphism we have $\mathrm{pres}_\Theta(\alpha_1) \leq \mathrm{pres}_\Theta(\alpha_2) \leq \cdots$. By definition $\lim_{n\to\infty} \mathrm{pres}_\Theta(\alpha_n) = \mathrm{pres}_\Theta(\alpha) = v \in \Theta^*$, from which it follows that there exists an $m \geq 1$ such that $\mathrm{pres}_\Theta(\alpha_m) = \mathrm{pres}_\Theta(\alpha_{m+k})$ for all $k \geq 0$. Hence $\mathrm{pres}_\Theta(\alpha) = \mathrm{pres}_\Theta(\alpha_m) \in \mathrm{pres}_\Theta(\mathbf{C}_{\mathcal{A}})$.

(2) ($\supseteq$) Immediate, by Definition 3.1.7(2,4).

($\subseteq$) Let $\alpha \in \mathbf{B}_{\mathcal{A}}^{\Theta,\omega}$. Then Definition 3.1.7(2,4) implies $\alpha \in \mathrm{pres}_{\Theta}(\mathbf{C}_{\mathcal{A}}^{\infty}) \cap \Theta^{\omega}$. Hence either $\alpha \in \mathrm{pres}_{\Theta}(\mathbf{C}_{\mathcal{A}}^{\omega}) \cap \Theta^{\omega}$ or $\alpha \in \mathrm{pres}_{\Theta}(\mathbf{C}_{\mathcal{A}}) \cap \Theta^{\omega} = \varnothing$. □

The finite computations thus determine the finitary behavior of an automaton. By Theorem 3.1.6, moreover, they also determine its infinitary behavior and thus the full behavior.

Theorem 3.1.10. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two automata and let Θ be an alphabet disjoint from their sets of states. Then*

if $\mathbf{C}_{\mathcal{A}} = \mathbf{C}_{\mathcal{A}'}$, *then* $\mathbf{B}_{\mathcal{A}}^{\Theta} = \mathbf{B}_{\mathcal{A}'}^{\Theta}$ *and* $\mathbf{B}_{\mathcal{A}}^{\Theta,\omega} = \mathbf{B}_{\mathcal{A}'}^{\Theta,\omega}$. □

Corollary 3.1.11. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two automata and let Θ be an alphabet disjoint from their sets of states. Then*

if $\mathbf{C}_{\mathcal{A}} = \mathbf{C}_{\mathcal{A}'}$, *then* $\mathbf{B}_{\mathcal{A}}^{\Theta,\infty} = \mathbf{B}_{\mathcal{A}'}^{\Theta,\infty}$. □

Unlike the situation for computations as formulated in Lemma 3.1.5 and Theorem 3.1.6, the finitary behavior of an automaton does not determine its infinitary behavior. The loss of information due to the omission of states prohibits combining “matching” finite records into an infinite record.

Example 3.1.12. Consider the two automata $\mathcal{A} = (Q, \{a\}, \delta, \{q\})$ and $\mathcal{A}' = (Q', \{a\}, \delta', \{q'\})$, where $Q = \{q, q_{11}, q_{21}, q_{22}, q_{31}, q_{32}, q_{33}, \ldots\}$, $Q' = \{q', q_1, q_2, q_3, \ldots\}$, and δ and δ' are as depicted in Figure 3.2.

It is easy to see that $\mathbf{C}_{\mathcal{A}}^{\omega} = \varnothing$, even though $\mathbf{C}_{\mathcal{A}} = \{q, qaq_{11}, qaq_{21}aq_{22}, \ldots\}$ is infinite. We furthermore see that $\mathbf{B}_{\mathcal{A}}^{\{a\}} = \mathbf{B}_{\mathcal{A}'}^{\{a\}} = \{\lambda, a, aa, aaa, \ldots\}$, whereas $a^{\omega} \in \mathbf{B}_{\mathcal{A}'}^{\{a\},\infty} \setminus \mathbf{B}_{\mathcal{A}}^{\{a\},\infty}$. In fact, $\mathbf{B}_{\mathcal{A}}^{\Sigma,\omega} = \varnothing$. □

By considering automata with a possibly infinite set of states we have chosen a computationally very powerful model. Any given Turing machine $\mathcal{M}$ can be unfolded into an automaton $\mathcal{A}$ that has the same behavior: $\mathcal{A}$ has all possible configurations of $\mathcal{M}$ as its set of states and a transition from a state C to C' with label p whenever $\mathcal{M}$ can move from configuration C to configuration C' by executing instruction p.

A direct consequence is that many problems or questions concerning automata that are decidable for finite automata are now undecidable, e.g., there exists no effective procedure for deciding for a given automaton whether or not a given state can be reached by a computation that starts from the initial state. If this problem would be decidable, then an effective decision procedure for the halting problem for Turing machines would exist, which is known to be undecidable.

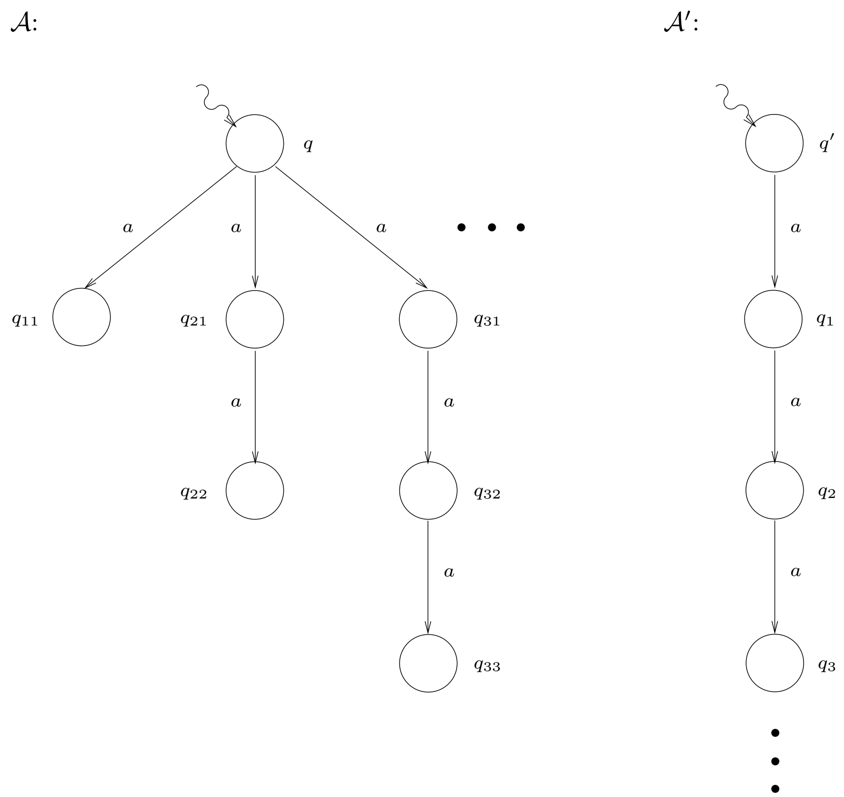


Fig. 3.2. Automata $\mathcal{A}$ and $\mathcal{A}'$.

3.2 Properties of Automata

In this section we discuss some basic notions for automata. In three subsections we consider reduced versions of automata, the enabling of actions in automata, and deterministic automata.

3.2.1 Reduced Versions

An automaton may have states, actions, or transitions that are "superfluous" in the sense that they do not occur in any computation of the automaton. Thus for the description and investigation of the dynamic — behavioral — properties of an automaton these elements are often not relevant and may be ignored.

In this subsection we introduce and relate to each other various reduced versions of an automaton. A reduced version of an automaton has less states, actions, or transitions than, but the same set of computations as, the original automaton.

We begin by identifying those elements of an automaton that are crucial for its set of computations and behavior, and which thus cannot be omitted from an automaton without affecting its set of computations and behavior.

Definition 3.2.1. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

(1) a state $q \in Q$ is reachable *(in $\mathcal{A}$) if there exists a computation $\alpha \in \mathbf{C}_{\mathcal{A}}^{\infty}$ such that $\alpha = \beta q \gamma$ for some $\beta \in (Q\Sigma)^*$ and $\gamma \in (\Sigma Q)^{\infty}$,*

(2) an action $a \in \Sigma$ is active *(in $\mathcal{A}$) if there exists a computation $\alpha \in \mathbf{C}_{\mathcal{A}}^{\infty}$ such that $\alpha = \beta a \gamma$ for some $\beta \in I(\Sigma Q)^*$ and $\gamma \in Q(\Sigma Q)^{\infty}$, and*

(3) a transition $(q, a, q') \in \delta$ is useful *(in $\mathcal{A}$) if there exists a computation $\alpha \in \mathbf{C}_{\mathcal{A}}^{\infty}$ such that $\alpha = \beta q a q' \gamma$ for some $\beta \in (Q\Sigma)^*$ and $\gamma \in (\Sigma Q)^{\infty}$.* □

By Definition 3.1.7, an action can occur in a (Θ-)record of an automaton if and only if it occurs in a computation of that automaton (and belongs to Θ). It thus suffices to focus on computations only and there is no need for an additional definition for actions occurring in the (Θ-)behavior of an automaton.

Every occurrence of a state in a computation marks the end of a finite computation (cf. the proof of Lemma 3.1.3). Thus a state is reachable if and only if it can be reached as a result of a finite computation. Recall that the initial states are always reachable by a trivial computation. Moreover, as an immediate consequence of their definitions, it follows that reachability of states, activity of actions, and usefulness of transitions can be established by following the paths laid out by the labeled transitions starting from initial states. However, one should keep in mind that — since no a priori constraints are imposed on the state space, the alphabet, and the set of transitions of an automaton — this is in general not an effective procedure.

Lemma 3.2.2. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

(1) a state $q \in Q$ is reachable in $\mathcal{A}$ if and only if there exists a finite computation $\alpha \in \mathbf{C}_{\mathcal{A}}$ such that $\alpha = \beta q$ for some $\beta \in (Q\Sigma)^*$,

(2) a transition $(q, a, q') \in \delta$ is useful in $\mathcal{A}$ if and only if q is reachable in $\mathcal{A}$,

(3) an action $a \in \Sigma$ is active in $\mathcal{A}$ if and only if there exists a useful transition $(q, a, q') \in \delta$, and

(4) if $(q, a, q') \in \delta$ is useful in $\mathcal{A}$, then q' is reachable in $\mathcal{A}$ and a is active in $\mathcal{A}$. □

Definition 3.2.3. *Let $\mathcal{A}$ be an automaton. Then*

(1) its set of reachable states is denoted by $Q_{\mathcal{A},S}$,

(2) its set of active actions is denoted by $\Sigma_{\mathcal{A},A}$, and

(3) its set of useful transitions is denoted by $\delta_{\mathcal{A},T}$. □

Whenever $\mathcal{A}$ is clear from the context, then we often simply use Q_S, Σ_A, and δ_T rather than $Q_{\mathcal{A},S}$, $\Sigma_{\mathcal{A},A}$, and $\delta_{\mathcal{A},T}$.

An immediate consequence of these definitions is the fact that the set of computations of an arbitrary automaton contains the set $\mathbf{C}_{\mathcal{A}}$ of computations of a given automaton $\mathcal{A}$, if and only if $Q_{\mathcal{A},S}$ is contained in its set of reachable states, $\Sigma_{\mathcal{A},A}$ is contained in its set of active actions, $\delta_{\mathcal{A},T}$ is contained in its set of useful transitions, and the initial states of $\mathcal{A}$ are among its initial states.

Lemma 3.2.4. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two automata with sets of initial states $I_{\mathcal{A}}$ and $I_{\mathcal{A}'}$, respectively. Then*

$\mathbf{C}_{\mathcal{A}} \subseteq \mathbf{C}_{\mathcal{A}'}$ *if and only if* $Q_{\mathcal{A},S} \subseteq Q_{\mathcal{A}',S}$, $\Sigma_{\mathcal{A},A} \subseteq \Sigma_{\mathcal{A}',A}$, $\delta_{\mathcal{A},T} \subseteq \delta_{\mathcal{A}',T}$, *and* $I_{\mathcal{A}} \subseteq I_{\mathcal{A}'}$. □

The reduced versions of automata we are about to define will again be automata. Since they are the result of omitting — and not of adding — certain elements, any reduced version of an automaton will always be *contained in* the original automaton in the following sense.

Definition 3.2.5. *Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, I_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, I_2)$ be two automata. Then*

$\mathcal{A}_1$ *is* contained in $\mathcal{A}_2$, *denoted by* $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$, *if* $Q_1 \subseteq Q_2$, $\Sigma_1 \subseteq \Sigma_2$, $\delta_1 \subseteq \delta_2$, *and* $I_1 \subseteq I_2$. □

The containment relation $\sqsubseteq$ is reflexive and transitive and hence a partial order on automata. Although it would be natural to say that $\mathcal{A}_1$ is a "subautomaton" of $\mathcal{A}_2$ whenever $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ holds, we refrain from doing so. The reason being that this might lead to confusion with the notion of subautomaton that we will introduce later in the context of synchronized automata.

Containment of one automaton in another implies that the first automaton has no other (initial) states, actions, or transitions than those already present in the second automaton. Consequently, it will also have no other computations.

Lemma 3.2.6. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata. Then* *if* $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$, *then* $\mathbf{C}_{\mathcal{A}_1} \subseteq \mathbf{C}_{\mathcal{A}_2}$. □

Note that by Lemma 3.1.5, $\mathbf{C}_{\mathcal{A}_1} \subseteq \mathbf{C}_{\mathcal{A}_2}$ implies $\mathbf{C}^{\omega}_{\mathcal{A}_1} \subseteq \mathbf{C}^{\omega}_{\mathcal{A}_2}$ and it thus suffices to refer to finite computations only.

Since an automaton may have states, actions, and transitions that never occur in its computations, this statement cannot be reversed unless the condition of containment is weakened by relating to initial states and useful transitions only.

Lemma 3.2.7. *Let* $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, I_1)$ *and* $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, I_2)$ *be two automata. Then*

$\mathbf{C}_{\mathcal{A}_1} \subseteq \mathbf{C}_{\mathcal{A}_2}$ *if and only if* $I_1 \subseteq I_2$ *and* $\delta_{\mathcal{A}_1,T} \subseteq \delta_2$. □

A reduced version $\mathcal{A}'$ of an automaton $\mathcal{A}$ lacks certain elements of $\mathcal{A}$, but should still define the same set of computations. Hence we require that $\mathcal{A}'$ is an automaton. Furthermore, from here on we will focus on finite computations. This is sufficient because according to Theorem 3.1.6 and Corollary 3.1.11, equality of the sets of finite computations of $\mathcal{A}$ and $\mathcal{A}'$ guarantees that also the sets of all computations of $\mathcal{A}$ and $\mathcal{A}'$ will be the same, as well as their Θ-behavior (for every set of actions Θ).

We distinguish three different criteria that can be used to reduce an automaton. We define separately reductions based on states, on actions, and on transitions, and subsequently we combine them. Action reductions and transition reductions are both described relative to a given set Θ of actions, similar to the definitions of the Θ-records and Θ-behavior of an automaton.

We begin by introducing the Θ-*action-reduced version of* an automaton $\mathcal{A}$, which is defined by omitting from the set of actions of $\mathcal{A}$ those actions from Θ that are not active in $\mathcal{A}$. Thus also the transitions of $\mathcal{A}$ which are labeled with an action from Θ that is not active in $\mathcal{A}$, will be omitted.

Definition 3.2.8. *Let* $\mathcal{A} = (Q, \Sigma, \delta, I)$ *be an automaton and let* Θ *be an alphabet disjoint from* Q. *Then*

(1) the Θ-action-reduced version of $\mathcal{A}$ *is the automaton denoted by* $\mathcal{A}^{\Theta}_{A}$ *and is defined as* $\mathcal{A}^{\Theta}_{A} = (Q, \Sigma^{\Theta}_{\mathcal{A},A}, \delta^{\Theta}_{\mathcal{A},A}, I)$, *where*

$$\Sigma^{\Theta}_{\mathcal{A},A} = \{a \in \Sigma \mid a \in \Theta \Rightarrow a \in \Sigma_{\mathcal{A},A}\} \text{ and}$$

$$\delta^{\Theta}_{\mathcal{A},A} = \delta \cap (Q \times \Sigma^{\Theta}_{\mathcal{A},A} \times Q), \text{ and}$$

(2) $\mathcal{A}$ *is* Θ-*action reduced if* $\mathcal{A} = \mathcal{A}^{\Theta}_{A}$. □

Whenever the automaton $\mathcal{A}$ is clear from the context, then we may simply write Σ_A^Θ and δ_A^Θ rather than $\Sigma_{\mathcal{A},A}^\Theta$ and $\delta_{\mathcal{A},A}^\Theta$, respectively.

Note that $\Sigma_A^\varnothing = \Sigma$ and $\Sigma_A^\Sigma = \Sigma_A$. In general, $\Sigma_A^\Theta = (\Sigma \setminus \Theta) \cup (\Sigma_A \cap \Theta)$. Observe furthermore that in δ_A^Θ there may still be transitions labeled with a symbol from Θ which are not useful in $\mathcal{A}$. We have $\delta_A^\Theta = \{(q,a,q') \in \delta \mid a \in \Theta \Rightarrow a \in \Sigma_A\}$. Hence $\delta_A^\varnothing = \delta$ and $\delta_A^\Sigma \supseteq \delta_T$. Consequently $\mathcal{A}_A^\varnothing = \mathcal{A}$, which shows that action reduction relative to $\varnothing$ does not affect the automaton.

Next we define the *Θ-transition-reduced version of* an automaton $\mathcal{A}$. Transitions that are labeled with an action from Θ are retained only if they are useful, while all other transitions remain.

Definition 3.2.9. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ be an alphabet disjoint from Q. Then*

(1) the Θ-transition-reduced version of $\mathcal{A}$ *is the automaton denoted by $\mathcal{A}_T^\Theta$ and is defined as $\mathcal{A}_T^\Theta = (Q, \Sigma, \delta_{\mathcal{A},T}^\Theta, I)$, where*

$$\delta_{\mathcal{A},T}^\Theta = \{(q,a,q') \in \delta \mid a \in \Theta \Rightarrow (q,a,q') \in \delta_{\mathcal{A},T}\}, \text{ and}$$

(2) $\mathcal{A}$ is Θ-transition reduced if $\mathcal{A} = \mathcal{A}_T^\Theta$. □

Whenever the automaton $\mathcal{A}$ is clear from the context, then we may simply write δ_T^Θ rather than $\delta_{\mathcal{A},T}^\Theta$.

Note that $\delta_T^\varnothing = \delta$ and thus $\mathcal{A}_T^\varnothing = \mathcal{A}$. Hence transition reduction relative to $\varnothing$ does not affect the automaton. Moreover, $\delta_T^\Sigma = \delta_T$ and — in general — $\delta_T^\Theta = (\delta \setminus (Q \times \Theta \times Q)) \cup (\delta_T \cap (Q \times \Theta \times Q))$. In fact, $\delta_T \subseteq \delta_T^\Theta \subseteq \delta_A^\Theta$. In the following example we show that both of these inclusions can be proper.

Example 3.2.10. Let $\mathcal{A} = (\{p,q\}, \{a,b\}, \delta, \{p\})$, with $\delta = \{(p,a,p), (q,a,q), (q,b,p)\}$, be an automaton. It is depicted in Figure 3.3(a).

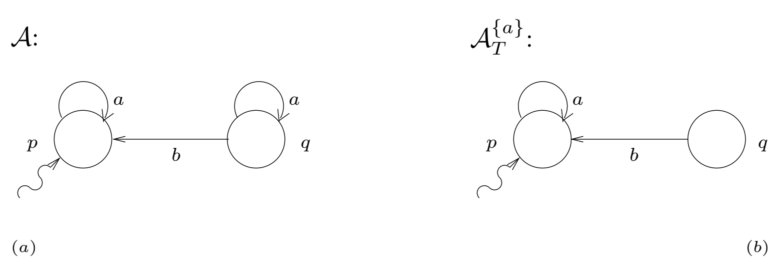


Fig. 3.3. Automata $\mathcal{A}$ and $\mathcal{A}_T^{\{a\}}$.

It is easy to see that $\delta_T = \{(p,a,p)\}$, i.e. $\mathcal{A}$ has only one useful transition. This implies that $\Sigma_A = \{a\}$ and thus $\delta_A^{\{a\}} = \delta$, i.e. $\mathcal{A}$ is $\{a\}$-action reduced: $\mathcal{A}_A^{\{a\}} = \mathcal{A}$. It also implies that the $\{a\}$-transition-reduced version of $\mathcal{A}$ is $\mathcal{A}_T^{\{a\}} = (\{p,q\},\{a,b\},\delta_T^{\{a\}},\{p\})$, with $\delta_T^{\{a\}} = \{(p,a,p),(q,b,p)\}$, as depicted in Figure 3.3(b). Consequently, $\delta_T \subsetneq \delta_T^{\{a\}} \subsetneq \delta_A^{\{a\}}$. □

Lemma 3.2.11. *Let $\mathcal{A} = (Q,\Sigma,\delta,I)$ be an automaton and let Θ be an alphabet disjoint from Q. Let $\mathcal{A}_A^\Theta = (Q,\Sigma_A^\Theta,\delta_A^\Theta,I)$ and let $\mathcal{A}_T^\Theta = (Q,\Sigma,\delta_T^\Theta,I)$. Then*

(1) $\delta_T = \delta_T^\Theta \setminus \{(q,a,q') \in \delta \mid a \notin \Theta$ and $(q,a,q') \notin \delta_T\}$ and

(2) $\delta_T^\Theta = \delta_A^\Theta \setminus \{(q,a,q') \in \delta \mid a \in \Theta$ and $(q,a,q') \notin \delta_T\}$.

Proof. (1) ($\subseteq$) Immediate because δ_T consists only of useful transitions.

($\supseteq$) This follows from the observation that all transitions $(q,a,q') \in \delta_T^\Theta$, with $a \in \Theta$, are useful in $\mathcal{A}$.

(2) ($\subseteq$) Let $(q,a,q') \in \delta_T^\Theta$. Thus $(q,a,q') \in \delta$.

If $a \notin \Theta$, then $a \in \Sigma_A^\Theta$ and so $(q,a,q') \in \delta_A^\Theta$.

If $a \in \Theta$, then $(q,a,q') \in \delta_T$.

Hence $(q,a,q') \in \delta_A^\Theta \setminus \{(q,a,q') \in \delta \mid a \in \Theta$ and $(q,a,q') \notin \delta_T\}$.

($\supseteq$) Let $(q,a,q') \in \delta_A^\Theta$ be such that $a \in \Theta$ implies $(q,a,q') \in \delta_T$. Then by Definition 3.2.9(1), $(q,a,q') \in \delta_T^\Theta$. □

It is immediate from the definitions that for every automaton $\mathcal{A}$ and for every set of actions Θ, both the Θ-action-reduced version $\mathcal{A}_A^\Theta$ of $\mathcal{A}$ and its Θ-transition-reduced version $\mathcal{A}_T^\Theta$ are contained in $\mathcal{A}$. Consequently, $\mathbf{C}_{\mathcal{A}_A^\Theta} \subseteq \mathbf{C}_\mathcal{A}$ and $\mathbf{C}_{\mathcal{A}_T^\Theta} \subseteq \mathbf{C}_\mathcal{A}$ always hold due to Lemma 3.2.6. In addition, Lemma 3.2.11 implies that the transition relations of both $\mathcal{A}_A^\Theta$ and $\mathcal{A}_T^\Theta$ contain δ_T. Since $\mathcal{A}_A^\Theta$ and $\mathcal{A}_T^\Theta$ have the same initial states as $\mathcal{A}$, it follows from Lemma 3.2.7 that $\mathbf{C}_\mathcal{A} \subseteq \mathbf{C}_{\mathcal{A}_A^\Theta}$ and $\mathbf{C}_\mathcal{A} \subseteq \mathbf{C}_{\mathcal{A}_T^\Theta}$.

We conclude that Definitions 3.2.8 and 3.2.9 thus satisfy the requirement that the computations of an automaton are not affected by the reduction.

Theorem 3.2.12. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

$$\mathbf{C}_\mathcal{A} = \mathbf{C}_{\mathcal{A}_A^\Theta} = \mathbf{C}_{\mathcal{A}_T^\Theta}.$$ □

An immediate consequence of this theorem is that an automaton, its Θ-action-reduced version, and its Θ-transition-reduced version, all three have the same reachable states, active actions, and useful transitions.

Corollary 3.2.13. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

(1) $Q_{\mathcal{A},S} = Q_{\mathcal{A}^{\Theta}_{A},S} = Q_{\mathcal{A}^{\Theta}_{T},S}$,

(2) $\Sigma_{\mathcal{A},A} = \Sigma_{\mathcal{A}^{\Theta}_{A},A} = \Sigma_{\mathcal{A}^{\Theta}_{T},A}$, *and*

(3) $\delta_{\mathcal{A},T} = \delta_{\mathcal{A}^{\Theta}_{A},T} = \delta_{\mathcal{A}^{\Theta}_{T},T}$. □

In Definitions 3.2.8 and 3.2.9, the reduced versions of an automaton are defined relative to some given alphabet Θ. From both definitions it is however immediately clear that actions which do belong to Θ but not to the alphabet of the automaton, are not even considered.

Lemma 3.2.14. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

(1) $\mathcal{A}^{\Theta}_{A} = \mathcal{A}^{\Theta}_{T} = \mathcal{A}$ *whenever* $\Theta \cap \Sigma = \varnothing$,

(2) $\mathcal{A}^{\Theta}_{A} = \mathcal{A}^{\Theta \cap \Sigma}_{A}$, *and*

(3) $\mathcal{A}^{\Theta}_{T} = \mathcal{A}^{\Theta \cap \Sigma}_{T}$. □

In addition, both in Definition 3.2.8 and in Definition 3.2.9 the role of each action is assessed on an individual basis, and reduction relative to any action is independent of the role of other actions.

Example 3.2.15. (Example 3.2.10 continued) Let $\mathcal{A}^2$ be the automaton obtained from $\mathcal{A}$ by adding the transition (p, c, p) to its transition relation. Then $\Sigma_{\mathcal{A}^2,A} = \{a, c\}$ are the active actions of $\mathcal{A}^2$. Hence $\mathcal{A}^2$ is $\{a\}$-action reduced, $\{c\}$-action reduced, and $\{a, c\}$-action reduced. Since b is not active in $\mathcal{A}^2$ it follows that $\mathcal{A}^2$ is neither $\{b\}$-action reduced, nor $\{a, b\}$-action reduced, nor $\{b, c\}$-action reduced.

The useful transitions of $\mathcal{A}^2$ are $\delta_{\mathcal{A}^2,T} = \{(p, a, p), (p, c, p)\}$. Hence $\mathcal{A}^2$ is not $\{a\}$-transition reduced as (q, a, q) is not useful in $\mathcal{A}^2$. Since also (q, b, p) is not useful in $\mathcal{A}^2$, it follows that $\mathcal{A}^2$ is neither $\{b\}$-transition reduced nor $\{a, b\}$-transition reduced. Because the only c-transition *is* useful in $\mathcal{A}^2$, we do have that $\mathcal{A}^2$ is $\{c\}$-transition reduced. However, $\mathcal{A}^2$ is neither $\{a, c\}$-transition reduced nor $\{b, c\}$-transition reduced. □

Consequently, as formally stated in the next lemma, the order in which actions are considered is irrelevant and has no effect on the resulting reduced version.

Lemma 3.2.16. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton, let Θ be an alphabet disjoint from Q, and let $\Theta_1, \Theta_2 \subseteq \Theta$ be such that $\Theta = \Theta_1 \cup \Theta_2$. Then*

(1) $(\mathcal{A}_A^{\Theta_1})_A^{\Theta_2} = \mathcal{A}_A^{\Theta}$ *and*

(2) $(\mathcal{A}_T^{\Theta_1})_T^{\Theta_2} = \mathcal{A}_T^{\Theta}$.

Proof. (1) Let $\mathcal{A}_A^{\Theta_1} = (Q, \Sigma_A^{\Theta_1}, \delta_A^{\Theta_1}, I)$, $(\mathcal{A}_A^{\Theta_1})_A^{\Theta_2} = (Q, (\Sigma_A^{\Theta_1})_A^{\Theta_2}, (\delta_A^{\Theta_1})_A^{\Theta_2}, I)$, and $\mathcal{A}_A^{\Theta_1 \cup \Theta_2} = \mathcal{A}_A^{\Theta} = (Q, \Sigma_A^{\Theta}, \delta_A^{\Theta}, I)$. First we prove that $(\Sigma_A^{\Theta_1})_A^{\Theta_2} = \Sigma_A^{\Theta}$.

Let $a \in (\Sigma_A^{\Theta_1})_A^{\Theta_2}$. Then $a \in \Sigma_A^{\Theta_1}$, which implies that $a \in \Sigma$.
If $a \notin \Theta$, then $a \in \Sigma_A^{\Theta}$ by definition.
If $a \in \Theta_1$, then $a \in \Sigma_{\mathcal{A},A}$ because $a \in \Sigma_A^{\Theta_1}$, and hence $a \in \Sigma_A^{\Theta}$.
If $a \in \Theta_2$, then $a \in \Sigma_{\mathcal{A}_A^{\Theta_1},A}$ because $a \in (\Sigma_A^{\Theta_1})_A^{\Theta_2}$. By Corollary 3.2.13 it follows that $a \in \Sigma_{\mathcal{A},A}$ and hence $a \in \Sigma_A^{\Theta}$.

Now assume that $a \in \Sigma_A^{\Theta}$. Then $a \in \Sigma$.
If $a \notin \Theta$, then by definition $a \in \Sigma_A^{\Theta_1}$ and $a \in (\Sigma_A^{\Theta_1})_A^{\Theta_2}$.
If $a \in \Theta$, then $a \in \Sigma_{\mathcal{A},A}$ because $a \in \Sigma_A^{\Theta}$ and by Corollary 3.2.13 also $a \in \Sigma_{\mathcal{A}_A^{\Theta_1},A}$. Hence $a \in \Sigma_A^{\Theta_1}$ and $a \in (\Sigma_A^{\Theta_1})_A^{\Theta_2}$.

Having established $(\Sigma_A^{\Theta_1})_A^{\Theta_2} = \Sigma_A^{\Theta}$ we immediately obtain that $(\delta_A^{\Theta_1})_A^{\Theta_2} = \delta_A^{\Theta_1} \cap (Q \times (\Sigma_A^{\Theta_1})_A^{\Theta_2} \times Q) = (\delta \cap (Q \times \Sigma_A^{\Theta_1} \times Q)) \cap (Q \times \Sigma_A^{\Theta} \times Q)$. Since $\Sigma_A^{\Theta} \subseteq \Sigma_A^{\Theta_1}$ this yields $(\delta_A^{\Theta_1})_A^{\Theta_2} = \delta \cap (Q \times \Sigma_A^{\Theta} \times Q) = \delta_A^{\Theta}$.

(2) Let $\mathcal{A}_T^{\Theta_1} = (Q, \Sigma, \delta_T^{\Theta_1}, I)$, let $(\mathcal{A}_T^{\Theta_1})_T^{\Theta_2} = (Q, \Sigma, (\delta_T^{\Theta_1})_T^{\Theta_2}, I)$, and let $\mathcal{A}_T^{\Theta_1 \cup \Theta_2} = \mathcal{A}_T^{\Theta} = (Q, \Sigma, \delta_T^{\Theta}, I)$. We prove that $(\delta_T^{\Theta_1})_T^{\Theta_2} = \delta_T^{\Theta}$.

Let $(q, a, q') \in (\delta_T^{\Theta_1})_T^{\Theta_2}$. Then $(q, a, q') \in \delta_T^{\Theta_1}$, which implies $(q, a, q') \in \delta$.
If $a \notin \Theta$, then $(q, a, q') \in \delta_T^{\Theta}$ by definition.
If $a \in \Theta_1$, then $(q, a, q') \in \delta_{\mathcal{A},T}$ because $(q, a, q') \in \delta_T^{\Theta_1}$, and hence $(q, a, q') \in \delta_T^{\Theta}$.
If $a \in \Theta_2$, then $(q, a, q') \in \delta_{\mathcal{A}_T^{\Theta_1},T}$ because $(q, a, q') \in (\delta_T^{\Theta_1})_T^{\Theta_2}$. By Corollary 3.2.13 it follows that $(q, a, q') \in \delta_{\mathcal{A},T}$ and hence $(q, a, q') \in \delta_T^{\Theta}$.

Now assume that $(q, a, q') \in \delta_T^{\Theta}$. Thus $(q, a, q') \in \delta$.
If $a \notin \Theta$, then by definition $(q, a, q') \in \delta_T^{\Theta_1}$ and $(q, a, q') \in (\delta_T^{\Theta_1})_T^{\Theta_2}$.
If $a \in \Theta$, then $(q, a, q') \in \delta_{\mathcal{A},T}$ because $(q, a, q') \in \delta_T^{\Theta}$. Thus by Corollary 3.2.13 we have $(q, a, q') \in \delta_{\mathcal{A}_T^{\Theta_1},T}$. Hence $(q, a, q') \in \delta_T^{\Theta_1}$ and $(q, a, q') \in (\delta_T^{\Theta_1})_T^{\Theta_2}$. □

An immediate consequence of this lemma is that the Θ-action-reduced and the Θ-transition-reduced versions of an automaton are indeed Θ-action-reduced and Θ-transition-reduced automata, respectively.

Theorem 3.2.17. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

(1) $\mathcal{A}_A^\Theta$ *is* Θ*-action reduced and*

(2) $\mathcal{A}_T^\Theta$ *is* Θ*-transition reduced.*

Proof. $\mathcal{A}_A^\Theta = (\mathcal{A}_A^\Theta)_A^\Theta$ and $\mathcal{A}_T^\Theta = (\mathcal{A}_T^\Theta)_T^\Theta$ follow directly from Lemma 3.2.16. □

A more general consequence is that reduction relative to more actions has a cumulative effect, but only for those actions that have not yet been considered there is an effect.

Lemma 3.2.18. *Let* $\mathcal{A} = (Q, \Sigma, \delta, I)$ *be an automaton and let* Θ_1, Θ_2 *be alphabets disjoint from* Q *and such that* $(\Theta_1 \cap \Sigma) \subseteq \Theta_2$*. Then*

(1) (i) $(\mathcal{A}_A^{\Theta_2})_A^{\Theta_1} = \mathcal{A}_A^{\Theta_2}$*,*

(ii) $\mathcal{A}_A^{\Theta_2} \sqsubseteq \mathcal{A}_A^{\Theta_1}$*, and*

(iii) if $\mathcal{A} = \mathcal{A}_A^{\Theta_2}$*, then* $\mathcal{A} = \mathcal{A}_A^{\Theta_1}$*, and*

(2) (i) $(\mathcal{A}_T^{\Theta_2})_T^{\Theta_1} = \mathcal{A}_T^{\Theta_2}$*,*

(ii) $\mathcal{A}_T^{\Theta_2} \sqsubseteq \mathcal{A}_T^{\Theta_1}$*, and*

(iii) if $\mathcal{A} = \mathcal{A}_T^{\Theta_2}$*, then* $\mathcal{A} = \mathcal{A}_T^{\Theta_1}$*.*

Proof. (1) (i) Let Σ' be the alphabet of $\mathcal{A}_A^{\Theta_2}$. Thus $\Sigma' \subseteq \Sigma$ and hence $\Theta_1 \cap \Sigma' \subseteq \Theta_1 \cap \Sigma \subseteq \Theta_2$. From Lemma 3.2.14(2) we know that $(\mathcal{A}_A^{\Theta_2})_A^{\Theta_1} = (\mathcal{A}_A^{\Theta_2})_A^{\Theta_1 \cap \Sigma'}$. Combining these facts with Lemma 3.2.16(1) yields $(\mathcal{A}_A^{\Theta_2})_A^{\Theta_1} = (\mathcal{A}_A^{\Theta_2})_A^{\Theta_1 \cap \Sigma'} = \mathcal{A}_A^{\Theta_2 \cup (\Theta_1 \cap \Sigma')} = \mathcal{A}_A^{\Theta_2}$.

(ii) Lemma 3.2.16(1) implies that $(\mathcal{A}_A^{\Theta_2})_A^{\Theta_1} = (\mathcal{A}_A^{\Theta_1})_A^{\Theta_2}$. Thus, by the above, $\mathcal{A}_A^{\Theta_2} = (\mathcal{A}_A^{\Theta_1})_A^{\Theta_2}$. Since reduction always yields an automaton contained in the original one, we now have $\mathcal{A}_A^{\Theta_2} = (\mathcal{A}_A^{\Theta_1})_A^{\Theta_2} \sqsubseteq \mathcal{A}_A^{\Theta_1}$.

(iii) Let $\mathcal{A} = \mathcal{A}_A^{\Theta_2}$. Then using (i) above we conclude that $\mathcal{A} = \mathcal{A}_A^{\Theta_2} = (\mathcal{A}_A^{\Theta_2})_A^{\Theta_1} = \mathcal{A}_A^{\Theta_1}$.

(2) (i) First we note that Σ is the alphabet of $\mathcal{A}_T^{\Theta_2}$. By Lemmata 3.2.13(3) and 3.2.16(2) we have $(\mathcal{A}_T^{\Theta_2})_T^{\Theta_1} = (\mathcal{A}_T^{\Theta_2})_T^{\Theta_1 \cap \Sigma} = \mathcal{A}_T^{\Theta_2 \cup (\Theta_1 \cap \Sigma)} = \mathcal{A}_T^{\Theta_2}$.

(ii) Lemma 3.2.16(1) implies that $(\mathcal{A}_T^{\Theta_2})_T^{\Theta_1} = (\mathcal{A}_T^{\Theta_1})_T^{\Theta_2}$. Then, by the above, $\mathcal{A}_T^{\Theta_2} = (\mathcal{A}_T^{\Theta_1})_T^{\Theta_2}$. Since the transition reductions always yield an automaton contained in the original one, we now have $\mathcal{A}_T^{\Theta_2} = (\mathcal{A}_T^{\Theta_1})_T^{\Theta_2} \sqsubseteq \mathcal{A}_T^{\Theta_1}$.

(iii) Let $\mathcal{A} = \mathcal{A}_T^{\Theta_2}$. Then from (2) (i) we conclude that $\mathcal{A} = \mathcal{A}_T^{\Theta_2} = (\mathcal{A}_T^{\Theta_2})_T^{\Theta_1} = \mathcal{A}_T^{\Theta_2}$. □

Since all actions of an automaton $\mathcal{A}$ with alphabet Σ have been considered, a further reduction with respect to actions of $\mathcal{A}_A^\Sigma$ or a further reduction with respect to transitions of $\mathcal{A}_T^\Sigma$ thus has no additional effect.

Theorem 3.2.19. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ be an alphabet disjoint from Q. Then*

(1) $\mathcal{A}_A^\Sigma \sqsubseteq \mathcal{A}_A^\Theta$ *and*

(2) $\mathcal{A}_T^\Sigma \sqsubseteq \mathcal{A}_T^\Theta$. □

From Lemma 3.2.6 it follows that whenever an automaton $\mathcal{A}_1$ is contained in an automaton $\mathcal{A}_2$, then all elements which are superfluous in $\mathcal{A}_2$ will certainly be superfluous in $\mathcal{A}_1$. This implies that action reduction and transition reduction are monotonous operations with respect to containment ($\sqsubseteq$).

Lemma 3.2.20. *Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, I_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, I_2)$ be two automata such that $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ and let Θ be an alphabet disjoint from $Q_1 \cup Q_2$. Then*

(1) $(\mathcal{A}_1)_A^\Theta \sqsubseteq (\mathcal{A}_2)_A^\Theta$ *and*

(2) $(\mathcal{A}_1)_T^\Theta \sqsubseteq (\mathcal{A}_2)_T^\Theta$.

Proof. (1) Let $(\mathcal{A}_1)_A^\Theta = (Q_1, (\Sigma_1)_A^\Theta, (\delta_1)_A^\Theta, I_1)$ and let $(\mathcal{A}_2)_A^\Theta = (Q_2, (\Sigma_2)_A^\Theta, (\delta_2)_A^\Theta, I_2)$. Since $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ we know that $Q_1 \subseteq Q_2$ and $I_1 \subseteq I_2$. By Lemma 3.2.6, $\mathbf{C}_{\mathcal{A}_1} \subseteq \mathbf{C}_{\mathcal{A}_2}$ and thus every action that is active in $\mathcal{A}_1$ is also active in $\mathcal{A}_2$. Hence $(\Sigma_1)_A^\Theta \subseteq (\Sigma_2)_A^\Theta$. This in turn implies that $(\delta_1)_A^\Theta \subseteq (\delta_2)_A^\Theta$ because the transition relation of $\mathcal{A}_1$ is contained in that of $\mathcal{A}_2$. We conclude that $(\mathcal{A}_1)_A^\Theta \sqsubseteq (\mathcal{A}_2)_A^\Theta$.

(2) Let $(\mathcal{A}_1)_T^\Theta = (Q_1, \Sigma_1, (\delta_1)_T^\Theta, I_1)$ and let $(\mathcal{A}_2)_T^\Theta = (Q_2, \Sigma_2, (\delta_2)_T^\Theta, I_2)$. Since $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ we know that $Q_1 \subseteq Q_2$, $\Sigma_1 \subseteq \Sigma_2$, and $I_1 \subseteq I_2$. From the fact that $\mathbf{C}_{\mathcal{A}_1} \subseteq \mathbf{C}_{\mathcal{A}_2}$ by Lemma 3.2.6, we deduce that every transition that is useful in $\mathcal{A}_1$ is useful also in $\mathcal{A}_2$. Hence $(\delta_1)_T^\Theta \subseteq (\delta_2)_T^\Theta$ and we conclude that $(\mathcal{A}_1)_T^\Theta \sqsubseteq (\mathcal{A}_2)_T^\Theta$. □

Given an alphabet Θ, an automaton $\mathcal{A}$ may contain many automata that are Θ-action reduced or Θ-transition reduced. We can now show that among these $\mathcal{A}_A^\Theta$ and $\mathcal{A}_T^\Theta$, respectively, are the largest (with respect to containment).

Lemma 3.2.21. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Let $\mathcal{A}' \sqsubseteq \mathcal{A}$. Then*

(1) if $\mathcal{A}'$ is Θ-action reduced, then $\mathcal{A}' \sqsubseteq \mathcal{A}_A^\Theta$, and

(2) if $\mathcal{A}'$ is Θ-transition reduced, then $\mathcal{A}' \sqsubseteq \mathcal{A}_T^\Theta$.

Proof. Since $\mathcal{A}' \sqsubseteq \mathcal{A}$, Lemma 3.2.20 implies $(\mathcal{A}')_A^\Theta \sqsubseteq \mathcal{A}_A^\Theta$ and $(\mathcal{A}')_T^\Theta \sqsubseteq \mathcal{A}_T^\Theta$. Hence if $\mathcal{A}' = (\mathcal{A}')_A^\Theta$, then $\mathcal{A}' \sqsubseteq \mathcal{A}_A^\Theta$, and if $\mathcal{A}' = (\mathcal{A}')_T^\Theta$, then $\mathcal{A}' \sqsubseteq \mathcal{A}_T^\Theta$. □

Theorem 3.2.22. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

(1) $\mathcal{A}_A^{\Theta}$ is the largest Θ-action-reduced automaton contained in $\mathcal{A}$ and

(2) $\mathcal{A}_T^{\Theta}$ is the largest Θ-transition-reduced automaton contained in $\mathcal{A}$.

Proof. Immediate from Theorem 3.2.17 and Lemma 3.2.21. □

For a given automaton $\mathcal{A}$ and an alphabet Θ, the difference between $\mathcal{A}$ and $\mathcal{A}_A^{\Theta}$ and between $\mathcal{A}$ and $\mathcal{A}_T^{\Theta}$ is thus minimal. Nevertheless, by definition, the remaining actions of Θ in $\mathcal{A}_A^{\Theta}$ are active in both $\mathcal{A}$ and $\mathcal{A}_A^{\Theta}$, and the remaining transitions in $\mathcal{A}_T^{\Theta}$ with a label from Θ are useful in both $\mathcal{A}$ and $\mathcal{A}_T^{\Theta}$. Hence, a further reduction of $\mathcal{A}_A^{\Theta}$ or $\mathcal{A}_T^{\Theta}$ that will not affect the computations is only feasible when other elements are considered. We already observed in Theorem 3.2.19 that in case all actions of $\mathcal{A}$ have been involved in action reduction (yielding $\mathcal{A}_A^{\Sigma}$) or transition reduction (yielding $\mathcal{A}_T^{\Sigma}$), further action reduction or transition reduction, respectively, will have no additional effect.

From Definitions 3.2.8 and 3.2.9 and the observations immediately following these definitions we know that given an automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$ we have $\mathcal{A}_A^{\Sigma} = (Q, \Sigma_{\mathcal{A},A}, \delta_A^{\Sigma}, I)$ and $\mathcal{A}_T^{\Sigma} = (Q, \Sigma, \delta_{\mathcal{A},T}, I)$, with $\Sigma_{\mathcal{A},A} \subseteq \Sigma$ and $\delta_{\mathcal{A},T} \subseteq \delta_A^{\Sigma}$. Hence $\mathcal{A}_A^{\Sigma}$ and $\mathcal{A}_T^{\Sigma}$ are in general incomparable. We now consider the effect of combining action and transition reductions.

Lemma 3.2.23. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ_1, Θ_2 be alphabets disjoint from Q. Then*

$$(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2} = (\mathcal{A}_T^{\Theta_2})_A^{\Theta_1}.$$

Proof. Let $\mathcal{A}_A^{\Theta_1} = (Q, \Sigma_A^{\Theta_1}, \delta_A^{\Theta_1}, I)$ and $\mathcal{A}_T^{\Theta_2} = (Q, \Sigma, \delta_T^{\Theta_2}, I)$. Then $(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2} = (Q, \Sigma_A^{\Theta_1}, \delta_2, I)$ with $\delta_2 = \{(q, a, q') \in \delta_A^{\Theta_1} \mid a \in \Theta_2 \Rightarrow (q, a, q') \in \delta_{\mathcal{A}_A^{\Theta_1},T}\}$. By Corollary 3.2.13(3), $(q, a, q') \in \delta_{\mathcal{A}_A^{\Theta_1},T}$ if and only if $(q, a, q') \in \delta_{\mathcal{A},T}$. Hence $\delta_2 = \{(q, a, q') \in \delta_A^{\Theta_1} \mid a \in \Theta_2 \Rightarrow (q, a, q') \in \delta_{\mathcal{A},T}\} = \delta_A^{\Theta_1} \cap \delta_T^{\Theta_2} = \delta_T^{\Theta_2} \cap (\delta \cap (Q \times \Sigma_A^{\Theta_1} \times Q))$. Since $\delta_T^{\Theta_2} \subseteq \delta$, we have $\delta_2 = \delta_T^{\Theta_2} \cap (Q \times \Sigma_A^{\Theta_1} \times Q)$.

Next consider $(\mathcal{A}_T^{\Theta_2})_A^{\Theta_1} = (Q, \Sigma_1, \delta_1, I)$, with $\Sigma_1 = \{a \in \Sigma \mid a \in \Theta_1 \Rightarrow a \in \Sigma_{\mathcal{A}_T^{\Theta_2},A}\}$ and $\delta_1 = \delta_T^{\Theta_2} \cap (Q \times \Sigma_1 \times Q)$. By Corollary 3.2.13(2), $a \in \Sigma_{\mathcal{A}_T^{\Theta_2},A}$ if and only if $a \in \Sigma_{\mathcal{A},A}$. Thus $\Sigma_1 = \{a \in \Sigma \mid a \in \Theta_1 \Rightarrow a \in \Sigma_{\mathcal{A},A}\} = \Sigma_A^{\Theta_1}$. Hence $\delta_1 = \delta_T^{\Theta_2} \cap (Q \times \Sigma_A^{\Theta_1} \times Q) = \delta_2$. We thus conclude that $(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2} = (\mathcal{A}_T^{\Theta_2})_A^{\Theta_1}$. □

By this lemma, the order in which action and transition reductions are applied is irrelevant. Together with Lemma 3.2.16 this implies that for every automaton $\mathcal{A}$, any finite succession of action reductions and transition reductions (relative to certain sets of actions) yields an automaton of the form $(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2} = (\mathcal{A}_T^{\Theta_2})_A^{\Theta_1}$.

Example 3.2.24. (Example 3.2.10 continued) We consider $\mathcal{A}$, as depicted in Figure 3.3(a). Since b is not active in $\mathcal{A}$, the $\{b\}$-action-reduced version of $\mathcal{A}$ is $\mathcal{A}_A^{\{b\}} = (\{p,q\},\{a\},\{(p,a,p),(q,a,q)\},\{p\})$. Because (q,a,q) is not useful in $\mathcal{A}_A^{\{b\}}$, the $\{a\}$-transition-reduced version of $\mathcal{A}_A^{\{b\}}$ is $(\mathcal{A}_A^{\{b\}})_T^{\{a\}} = (\{p,q\},\{a\},\{(p,a,p)\},\{p\})$.

Now we consider the $\{a\}$-transition-reduced version $\mathcal{A}_T^{\{a\}}$ of $\mathcal{A}$, as depicted in Figure 3.3(b). Since b is not active in $\mathcal{A}_T^{\{a\}}$, the $\{b\}$-action-reduced version of $\mathcal{A}_T^{\{a\}}$ is $(\mathcal{A}_T^{\{a\}})_A^{\{b\}} = (\mathcal{A}_A^{\{b\}})_T^{\{a\}}$. □

Theorem 3.2.25. *Let $\mathcal{A}$ be an automaton and let Θ_1, Θ_2 be alphabets disjoint from its set of states. Then*

(1) $(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2}$ is the largest automaton contained in $\mathcal{A}$ that is both Θ_1-action reduced and Θ_2-transition reduced, and

(2) $\mathbf{C}_{(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2}} = \mathbf{C}_\mathcal{A}$.

Proof. (1) By Lemma 3.2.23, $(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2} = (\mathcal{A}_T^{\Theta_2})_A^{\Theta_1}$. Using Lemma 3.2.16 it is easy to see that $(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2}$ is both Θ_1-action reduced and Θ_2-transition reduced. Now let $\mathcal{A}_1$ be an automaton contained in $\mathcal{A}$. Then, by Lemma 3.2.20, $(\mathcal{A}_1)_A^{\Theta_1} \sqsubseteq \mathcal{A}_A^{\Theta_1}$ and thus $((\mathcal{A}_1)_A^{\Theta_1})_T^{\Theta_2} \sqsubseteq (\mathcal{A}_A^{\Theta_1})_T^{\Theta_2}$. If $\mathcal{A}_1$ is Θ_1-action reduced and Θ_2-transition reduced, then $\mathcal{A}_1 = (\mathcal{A}_1)_A^{\Theta_1}$ and $\mathcal{A}_1 = (\mathcal{A}_1)_T^{\Theta_2}$. In that case we have $\mathcal{A}_1 = (\mathcal{A}_1)_A^{\Theta_1} = ((\mathcal{A}_1)_A^{\Theta_1})_T^{\Theta_2} \sqsubseteq (\mathcal{A}_A^{\Theta_1})_T^{\Theta_2}$.

(2) From Theorem 3.2.12 directly follows $\mathbf{C}_{(\mathcal{A}_A^{\Theta_1})_T^{\Theta_2}} = \mathbf{C}_{\mathcal{A}_A^{\Theta_1}} = \mathbf{C}_\mathcal{A}$. □

In particular we now have that given an automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$, the two automata $(\mathcal{A}_A^\Sigma)_T^\Sigma$ and $(\mathcal{A}_T^\Sigma)_A^\Sigma$ are the same. In fact, the definitions together with Theorem 3.2.12 and Corollary 3.2.13 imply that $(\mathcal{A}_A^\Sigma)_T^\Sigma = (Q, \Sigma_{\mathcal{A},A}, \delta_{\mathcal{A},T}, I) = (\mathcal{A}_T^\Sigma)_A^\Sigma$ and this automaton has neither superfluous actions nor superfluous transitions.

Theorem 3.2.26. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

(1) $\mathcal{A}_T^\Sigma$ is the least automaton with set of states Q and alphabet Σ such that $\mathbf{C}_{\mathcal{A}_T^\Sigma} = \mathbf{C}_\mathcal{A}$, and

(2) $(\mathcal{A}_A^\Sigma)_T^\Sigma$ is the least automaton with set of states Q such that $\mathbf{C}_{(\mathcal{A}_A^\Sigma)_T^\Sigma} = \mathbf{C}_\mathcal{A}$.

Proof. By Theorem 3.2.12, $\mathbf{C}_{\mathcal{A}_T^\Sigma} = \mathbf{C}_\mathcal{A} = \mathbf{C}_{\mathcal{A}_A^\Sigma} = \mathbf{C}_{(\mathcal{A}_A^\Sigma)_T^\Sigma}$. As observed before, $\mathcal{A}_T^\Sigma = (Q, \Sigma, \delta_{\mathcal{A},T}, I)$ and $(\mathcal{A}_A^\Sigma)_T^\Sigma = (Q, \Sigma_{\mathcal{A},A}, \delta_{\mathcal{A},T}, I)$. Now assume that $\mathcal{A}' = (Q, \Sigma', \delta', I')$ is an automaton such that $\mathbf{C}_{\mathcal{A}'} = \mathbf{C}_\mathcal{A}$. Thus $I' = I$, $\delta_{\mathcal{A}',T} = \delta_{\mathcal{A},T}$, and $\Sigma_{\mathcal{A}',A} = \Sigma_{\mathcal{A},A}$. Since $\delta_{\mathcal{A}',T} \subseteq \delta'$ and $\Sigma_{\mathcal{A}',A} \subseteq \Sigma'$ we have $(\mathcal{A}_A^\Sigma)_T^\Sigma \sqsubseteq \mathcal{A}'$, and if $\Sigma' = \Sigma$, then we have $\mathcal{A}_T^\Sigma \sqsubseteq \mathcal{A}'$. □

Finally, we consider (additional) reductions with respect to states.

The state-reduced version of an automaton is defined by omitting the non-reachable states from its specification. Consequently, the outgoing and incoming transitions of these states are no longer proper transitions and thus disappear as well.

Definition 3.2.27. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

(1) the state-reduced version of $\mathcal{A}$ *is the automaton denoted by $\mathcal{A}_S$ and is defined as $\mathcal{A}_S = (Q_S, \Sigma, \delta_T, I)$, and*

(2) $\mathcal{A}$ is state reduced if $\mathcal{A} = \mathcal{A}_S$. □

Note that $\delta_T = \{(q, a, q') \in \delta \mid q, q' \in Q_S\}$ by Lemma 3.2.2. Exactly those transitions that are outgoing or incoming transitions of a non-reachable state of $\mathcal{A}$ have thus been omitted. Hence $\delta_T = \delta \cap (Q_S \times \Sigma \times Q_S)$ and, since $I \subseteq Q_S$, $\mathcal{A}_S$ is well defined. Now Lemma 3.2.7 immediately implies that $\mathbf{C}_\mathcal{A} \subseteq \mathbf{C}_{\mathcal{A}_S}$. Furthermore, since $\mathcal{A}_S \sqsubseteq \mathcal{A}$ we know from Lemma 3.2.6 that $\mathbf{C}_{\mathcal{A}_S} \subseteq \mathbf{C}_\mathcal{A}$.

Theorem 3.2.28. *Let $\mathcal{A}$ be an automaton. Then*

$$\mathbf{C}_\mathcal{A} = \mathbf{C}_{\mathcal{A}_S}. \quad \square$$

Example 3.2.29. (Example 3.2.10 continued) Consider the automaton $\mathcal{A}$ depicted in Figure 3.3(a). We have seen that $\delta_T = \{(p, a, p)\}$. This implies that $Q_S = \{p\}$. Hence the state-reduced version of $\mathcal{A}$ is $\mathcal{A}_S = (\{p\}, \{a, b\}, \{(p, a, p)\}, \{p\})$ and thus $\mathbf{C}_\mathcal{A} = \mathbf{C}_{\mathcal{A}_S} = \{p, pap, papap, \dots\}$. □

Using the notion of a state-reduced version we can now reformulate Lemmata 3.2.6 and 3.2.7.

Lemma 3.2.30. *Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, I_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, I_2)$ be two automata such that $\Sigma_1 \subseteq \Sigma_2$. Then*

$\mathbf{C}_{\mathcal{A}_1} \subseteq \mathbf{C}_{\mathcal{A}_2}$ *if and only if* $(\mathcal{A}_1)_S \sqsubseteq (\mathcal{A}_2)_S$.

Proof. (Only if) Let $\mathbf{C}_{\mathcal{A}_1} \subseteq \mathbf{C}_{\mathcal{A}_2}$. Then by Lemma 3.2.7, $I_1 \subseteq I_2$ and $\delta_{\mathcal{A}_1,T} \subseteq \delta_2$. In fact, $\delta_{\mathcal{A}_1,T} \subseteq \delta_{\mathcal{A}_2,T}$ holds because all transitions in $\delta_{\mathcal{A}_1,T}$ are used in the computations of $\mathcal{A}_2$. From $\delta_{\mathcal{A}_1,T} \subseteq \delta_{\mathcal{A}_2,T}$ and Lemma 3.2.2 now follows that we also have $Q_{\mathcal{A}_1,S} \subseteq Q_{\mathcal{A}_2,S}$. Together with the fact that $\Sigma_1 \subseteq \Sigma_2$ this proves that $(\mathcal{A}_1)_S \sqsubseteq (\mathcal{A}_2)_S$.

(If) Let $(\mathcal{A}_1)_S \sqsubseteq (\mathcal{A}_2)_S$. Then $\mathbf{C}_{\mathcal{A}_1} = \mathbf{C}_{(\mathcal{A}_1)_S} \subseteq \mathbf{C}_{(\mathcal{A}_2)_S} = \mathbf{C}_{\mathcal{A}_2}$ by Lemma 3.2.6 and Theorem 3.2.28. □

As a consequence we obtain that also state reduction is a monotonous operation with respect to containment ($\sqsubseteq$).

Lemma 3.2.31. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata such that $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$. Then*

$(\mathcal{A}_1)_S \sqsubseteq (\mathcal{A}_2)_S$.

Proof. By Lemma 3.2.6, $\mathbf{C}_{\mathcal{A}_1} \subseteq \mathbf{C}_{\mathcal{A}_2}$, and since the alphabet of $\mathcal{A}_1$ is contained in that of $\mathcal{A}_2$, Lemma 3.2.30 implies that $(\mathcal{A}_1)_S \sqsubseteq (\mathcal{A}_2)_S$. □

Another consequence of Lemma 3.2.30 is that once an automaton has been reduced with respect to its states, no further state reduction is possible.

Theorem 3.2.32. *Let $\mathcal{A}$ be an automaton. Then*

$\mathcal{A}_S$ is state reduced.

Proof. By definition, $\mathcal{A}$ and $\mathcal{A}_S$ have the same alphabet. By Theorem 3.2.28, $\mathbf{C}_{\mathcal{A}} = \mathbf{C}_{\mathcal{A}_S}$. Since $\mathcal{A}$ and $\mathcal{A}_S$ have the same alphabet we can now apply Lemma 3.2.30 twice and thus obtain $\mathcal{A} = (\mathcal{A}_S)_S$. Consequently, $\mathcal{A}_S$ is state reduced. □

A state-reduced version of an automaton has neither superfluous states nor superfluous transitions.

Theorem 3.2.33. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

$\mathcal{A}_S$ is the least automaton with alphabet Σ such that $\mathbf{C}_{\mathcal{A}_S} = \mathbf{C}_{\mathcal{A}}$.

Proof. By definition, $\mathcal{A}_S$ and $\mathcal{A}$ have the same alphabet. By Theorem 3.2.28, $\mathbf{C}_{\mathcal{A}_S} = \mathbf{C}_{\mathcal{A}}$. Now assume that $\mathcal{A}'$ is an automaton with alphabet Σ and such that $\mathbf{C}_{\mathcal{A}} = \mathbf{C}_{\mathcal{A}'}$. Then by applying Lemma 3.2.30 twice we have $\mathcal{A}_S = (\mathcal{A}')_S \sqsubseteq \mathcal{A}'$. □

Though an automaton $\mathcal{A}$ may still contain many automata that are state reduced, we now show that among these $\mathcal{A}_S$ is the largest (with respect to containment).

Lemma 3.2.34. *Let $\mathcal{A}$ be an automaton and let $\mathcal{A}' \sqsubseteq \mathcal{A}$. Then*

if $\mathcal{A}'$ is state reduced, then $\mathcal{A}' \sqsubseteq \mathcal{A}_S$.

Proof. If $\mathcal{A}' = (\mathcal{A}')_S$, then by Lemma 3.2.31, $\mathcal{A}' = (\mathcal{A}')_S \sqsubseteq \mathcal{A}_S$. □

The difference between $\mathcal{A}$ and $\mathcal{A}_S$ is thus minimal.

Theorem 3.2.35. *Let $\mathcal{A}$ be an automaton. Then*

$\mathcal{A}_S$ is the largest state-reduced automaton contained in $\mathcal{A}$.

Proof. Immediate from Theorem 3.2.32 and Lemma 3.2.34. □

A further reduction can only be achieved through the actions and transitions. We thus combine state reductions with action reductions and transition reductions.

Lemma 3.2.36. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ be an alphabet disjoint from Q. Then*

(1) $(\mathcal{A}^{\Theta}_{A})_S = (\mathcal{A}_S)^{\Theta}_{A}$ and

(2) $(\mathcal{A}^{\Theta}_{T})_S = (\mathcal{A}_S)^{\Theta}_{T} = \mathcal{A}_S$.

Proof. (1) Let $\mathcal{A}^{\Theta}_{A} = (Q, \Sigma^{\Theta}_{A}, \delta^{\Theta}_{A}, I)$. By Corollary 3.2.13, $Q_{\mathcal{A}^{\Theta}_{A},S} = Q_{\mathcal{A},S}$ and $\delta_{\mathcal{A}^{\Theta}_{A},T} = \delta_{\mathcal{A},T}$. Hence $(\mathcal{A}^{\Theta}_{A})_S = (Q_{\mathcal{A},S}, \Sigma^{\Theta}_{A}, \delta_{\mathcal{A},T}, I)$.

Next we consider $(\mathcal{A}_S)^{\Theta}_{A} = (Q', \Sigma', \delta', I')$. By Definitions 3.2.8 and 3.2.27, $I' = I$ and $Q' = Q_{\mathcal{A},S}$. Furthermore, $\Sigma' = \{a \in \Sigma \mid a \in \Theta \Rightarrow a \in \Sigma_{\mathcal{A}_S,A}\}$. Since $\mathbf{C}_{\mathcal{A}_S} = \mathbf{C}_{\mathcal{A}}$ by Theorem 3.2.28, we have $\Sigma' = \{a \in \Sigma \mid a \in \Theta \Rightarrow \Sigma_{\mathcal{A},A}\} = \Sigma^{\Theta}_{A}$. Finally, $\delta' = \delta_{\mathcal{A},T} \cap (Q \times \Sigma^{\Theta}_{A} \times Q) = \delta_{\mathcal{A},T}$. Hence $(\mathcal{A}^{\Theta}_{A})_S = (\mathcal{A}_S)^{\Theta}_{A}$.

(2) Both $\mathcal{A}$ and $\mathcal{A}^{\Theta}_{T}$ have alphabet Σ. By Theorem 3.2.12, $\mathbf{C}_{\mathcal{A}} = \mathbf{C}_{\mathcal{A}^{\Theta}_{T}}$ and thus applying Lemma 3.2.30 twice yields $\mathcal{A}_S = (\mathcal{A}^{\Theta}_{T})_S$. Also $\mathcal{A}$ and $(\mathcal{A}_S)^{\Theta}_{T}$ have the same alphabet. Since $\mathbf{C}_{\mathcal{A}} = \mathbf{C}_{(\mathcal{A}_S)^{\Theta}_{T}}$ by Theorems 3.2.12 and 3.2.28, applying Lemma 3.2.30 twice yields $\mathcal{A}_S = ((\mathcal{A}_S)^{\Theta}_{T})_S$. Thus $\mathcal{A}_S = ((\mathcal{A}_S)^{\Theta}_{T})_S \sqsubseteq (\mathcal{A}_S)^{\Theta}_{T} \sqsubseteq \mathcal{A}_S$ and hence it must be the case that $\mathcal{A}_S = (\mathcal{A}_S)^{\Theta}_{T}$. □

Transition reduction in the context of state reduction thus has no effect. All transitions that are not useful will disappear by the state reduction.

Theorem 3.2.37. *Let $\mathcal{A}$ be a state-reduced automaton and let Θ be an alphabet disjoint from its set of states. Then*

$\mathcal{A}$ is Θ-transition reduced.

Proof. Since $\mathcal{A}$ is state reduced we have $\mathcal{A} = \mathcal{A}_S$. Then Lemma 3.2.36(2) implies $\mathcal{A}_T^\Theta = (\mathcal{A}_S)_T^\Theta = \mathcal{A}_S = \mathcal{A}$ and hence $\mathcal{A}$ is Θ-transition reduced. $\square$

Example 3.2.38. (Example 3.2.29 continued) By definition every transition of $\mathcal{A}_S$ is useful. Hence $\mathcal{A}_S$ trivially is Θ-transition reduced for any set of actions Θ. $\square$

Lemmata 3.2.16, 3.2.23, and 3.2.36 now imply that for every automaton $\mathcal{A}$, any finite succession of action reductions and state reductions (at least one) has the same effect as one state reduction and one action reduction (relative to some alphabet Θ) and yields an automaton $(\mathcal{A}_A^\Theta)_S = (\mathcal{A}_S)_A^\Theta$.

Example 3.2.39. (Examples 3.2.24 and 3.2.29 continued) Consider the state-reduced version $\mathcal{A}_S$ of $\mathcal{A}$. Since $\Sigma_{\mathcal{A}_S,A} = \{a\}$, the $\{b\}$-action-reduced version of $\mathcal{A}_S$ is $(\mathcal{A}_S)_A^{\{b\}} = (\{p\}, \{a\}, \{(p, a, p)\}, \{p\})$.

Now consider the $\{b\}$-action-reduced version $\mathcal{A}_A^{\{b\}}$ of $\mathcal{A}$. We have seen that its only useful transition is (p, a, p), which implies that q is not reachable and thus $(\mathcal{A}_A^{\{b\}})_S = (\mathcal{A}_S)_A^{\{b\}}$. $\square$

Theorem 3.2.40. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

> $(\mathcal{A}_A^\Theta)_S$ *is the largest automaton contained in $\mathcal{A}$ that is both state reduced and Θ-action reduced.*

Proof. By Lemma 3.2.36(1) and Theorems 3.2.17(1) and 3.2.32, $(\mathcal{A}_A^\Theta)_S = (\mathcal{A}_S)_A^\Theta$ is Θ-action reduced and state reduced.

Now let $\mathcal{A}_1 \sqsubseteq \mathcal{A}$. Then by Lemma 3.2.20(1), $(\mathcal{A}_1)_A^\Theta \sqsubseteq \mathcal{A}_A^\Theta$, and by Lemma 3.2.31, $((\mathcal{A}_1)_A^\Theta)_S \sqsubseteq (\mathcal{A}_A^\Theta)_S$. If $\mathcal{A}_1$ is Θ-action reduced, then by definition $(\mathcal{A}_1)_A^\Theta = \mathcal{A}_1$. If — in addition — it is state reduced, then $\mathcal{A}_1 = (\mathcal{A}_1)_S = ((\mathcal{A}_1)_A^\Theta)_S \sqsubseteq (\mathcal{A}_A^\Theta)_S$. $\square$

Summarizing, an automaton may have superfluous states, actions, or transitions, which can be omitted without affecting its operational potential (as represented by its set of finite computations). We have considered reductions with respect to each of these elements separately, and in combination. It has been shown that transition reduction is implied by state reduction, whereas the other combinations of reductions are stronger than each reduction separately. Consequently, once an automaton has been reduced with respect to states and actions, then it cannot be reduced any further without losing computations.

In correspondence to the notions of Θ-records and Θ-behavior of an automaton, both action reduction and transition reduction have been investigated relative to an alphabet. In case no special actions are distinguished and every element of the alphabet of an automaton is considered, then we drop in the sequel — as before — the reference to the alphabet if this cannot lead to confusion.

The above implies that for an automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$ we now have $\mathcal{A}_A = \mathcal{A}_A^\Sigma$ as its *action-reduced version*, and we have $\mathcal{A}_T = \mathcal{A}_T^\Sigma$ as its *transition-reduced version*. Furthermore, we will refer to $\mathcal{A}_R = (\mathcal{A}_A)_S = (\mathcal{A}_S)_A$ as the *reduced version of* $\mathcal{A}$. Note that the definitions of $\mathcal{A}_S$ and $(\mathcal{A}_S)_A^\Sigma$, together with Theorem 3.2.28 and Corollary 3.2.13, imply that the automaton $\mathcal{A}_R$ is specified as $\mathcal{A}_R = (Q_S, \Sigma_A, \delta_T, I)$. Hence $\mathcal{A}_R$ has no superfluous elements at all.

Theorems 3.2.37 and 3.2.40 imply that $\mathcal{A}_R$ is the largest automaton contained in $\mathcal{A}$ that is state reduced, action reduced, and transition reduced, and has the same computations as $\mathcal{A}$. We now show that $\mathcal{A}_R$ is the only such automaton.

Theorem 3.2.41. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

> *$\mathcal{A}_R$ is the unique automaton contained in $\mathcal{A}$ that is state reduced, action reduced, and transition reduced, and such that $\mathbf{C}_{\mathcal{A}_R} = \mathbf{C}_{\mathcal{A}}$.*

Proof. Let $\mathcal{A}' = (Q', \Sigma', \delta', I')$ be an action-reduced, transition-reduced, and state-reduced automaton such that $\mathcal{A}' \sqsubseteq \mathcal{A}$. From Theorems 3.2.37 and 3.2.40 we know that $\mathcal{A}' \sqsubseteq \mathcal{A}_R$.

Now assume that $\mathbf{C}_{\mathcal{A}'} = \mathbf{C}_{\mathcal{A}}$. Then $Q_{\mathcal{A}',S} = Q_{\mathcal{A},S}$, $\Sigma_{\mathcal{A}',A} = \Sigma_{\mathcal{A},A}$, $\delta_{\mathcal{A}',T} = \delta_{\mathcal{A},T}$, and $I' = I$. Since $Q_{\mathcal{A}',S} \subseteq Q'$, $\Sigma_{\mathcal{A}',A} \subseteq \Sigma'$, and $\delta_{\mathcal{A}',T} \subseteq \delta'$, we have $\mathcal{A}_R = (Q_{\mathcal{A},S}, \Sigma_{\mathcal{A},A}, \delta_{\mathcal{A},T}, I) \sqsubseteq \mathcal{A}'$. We thus conclude that $\mathcal{A}' = \mathcal{A}_R$. □

3.2.2 Enabling

For an arbitrary automaton and a given action, it is in general not the case that this action can always (i.e. at any give state) be executed by the automaton. For certain types of systems (such as, e.g., reactive systems) it may however be crucial that specific actions (in reaction to stimuli from the environment) can always be executed. Thus when such a system is modeled as an automaton, the transition relation should contain a transition for each of these actions at every (reachable) state.

In this subsection, we define *enabledness* of actions as a local (state dependent) property of the transition relation and then lift it to the level of the automaton. This contrasts with our approach in the previous subsection in which the role of states, actions, and transitions was assessed on basis of their occurrence in computations.

Definition 3.2.42. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

(1) an action $a \in \Sigma$ is enabled (in $\mathcal{A}$) at *a state $q \in Q$, denoted by $a \ en_{\mathcal{A}} \ q$, if $(q, a, q') \in \delta$ for some $q' \in Q$.*

Let Θ be an alphabet disjoint from Q. Then

*(2) $\mathcal{A}$ is Θ-*enabling *if for all $a \in \Theta$ and for all $q \in Q$, $a \in \Sigma \Rightarrow a \ en_{\mathcal{A}} \ q$.* □

Note that, as in previous definitions, also the property of enabling is defined with respect to a separately specified arbitrary set of actions Θ. Similar to those previous notions, whether or not an automaton is Θ-enabling is solely determined by those elements of Θ that are actions of $\mathcal{A}$. To be precise, $\mathcal{A}$ is always $\varnothing$-enabling. Furthermore, $\mathcal{A}$ is Θ-enabling if and only if it is $\Theta \cap \Sigma$-enabling, where Σ is the set of actions of $\mathcal{A}$.

Example 3.2.43. (Example 3.2.10 continued) It is easy to see that $\mathcal{A}$ is $\{a\}$-enabling but not $\{b\}$-enabling. Hence $\mathcal{A}$ is neither $\{a, b\}$-enabling. However, $\mathcal{A}$ is $\{d\}$-enabling, for all $d \notin \Sigma$, and thus also $\{a, d\}$-enabling. □

The deletion of states and/or transitions from an automaton does not affect its enabling of given actions, provided relevant transitions are preserved.

Lemma 3.2.44. *Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, I_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, I_2)$ be two automata and let Θ_1, Θ_2 be two alphabets disjoint from $Q_1 \cup Q_2$. Let $Q_2 \subseteq Q_1$, $\Theta_2 \cap \Sigma_2 \subseteq \Theta_1 \cap \Sigma_1$, and $\delta_2 \supseteq \delta_1 \cap (Q_2 \times (\Theta_2 \cap \Sigma_2) \times Q_1)$. Then*

if $\mathcal{A}_1$ is Θ_1-enabling, then $\mathcal{A}_2$ is Θ_2-enabling.

Proof. Let $\mathcal{A}_1$ be Θ_1-enabling. Now let $a \in \Theta_2$ and let $q \in Q_2$. If $a \in \Sigma_2$, then $a \in \Theta_1 \cap \Sigma_1$. Since $q \in Q_1$, it then follows that there exists a $q' \in Q$ such that $(q, a, q') \in \delta_1$. Thus $(q, a, q') \in \delta_2$ and we have a en$_{\mathcal{A}_2}$ q. □

Corollary 3.2.45. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ_1, Θ_2 be two alphabets disjoint from Q and such that $(\Theta_2 \cap \Sigma) \subseteq \Theta_1$. Then*

if $\mathcal{A}$ is Θ_1-enabling, then $\mathcal{A}$ is Θ_2-enabling. □

From the computational and the behavioral point of view, enabledness of actions is especially relevant at the reachable states of an automaton. Recall that for a given automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$ we denote by Q_S its set of reachable states. We have defined $\mathcal{A}_S = (Q_S, \Sigma, \delta_T, I)$ as the state-reduced version of $\mathcal{A}$, where $\delta_T = \delta \cap (Q_S \times \Sigma \times Q_S) = \delta \cap (Q_S \times \Sigma \times Q)$ consists of the useful transitions of $\mathcal{A}$. Thus, as another immediate consequence of Lemma 3.2.44, we have that the state-reduced version of $\mathcal{A}$ is Θ-enabling whenever $\mathcal{A}$ is.

Theorem 3.2.46. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

if $\mathcal{A}$ is Θ-enabling, then $\mathcal{A}_S$ is Θ-enabling. □

The converse clearly does not hold, since actions which are enabled at reachable states of an automaton $\mathcal{A}$ are not necessarily enabled at every non-reachable state of $\mathcal{A}$. The fact that the state-reduced version of $\mathcal{A}$ may have less states than $\mathcal{A}$ thus causes a lack of information concerning outgoing transitions of non-reachable states.

The situation is different when $\mathcal{A}$ is reduced by removing only its non-useful transitions with a label from an alphabet Θ_1, but no states whatsoever, as is done in order to obtain its Θ_1-transition-reduced version $\mathcal{A}_T^{\Theta_1}$. In that case the enabledness of actions in $\mathcal{A}_T^{\Theta_1}$ *can* thus be used to decide their enabledness in $\mathcal{A}$. In fact, since $\mathcal{A}_T^{\Theta_1}$ may have less transitions than $\mathcal{A}$, but it may never have less states than $\mathcal{A}$, Lemma 3.2.44 immediately yields the following result.

Lemma 3.2.47. *Let $\mathcal{A}$ be an automaton and let Θ, Θ_1 be two alphabets disjoint from its set of states. Then*

if $\mathcal{A}_T^{\Theta_1}$ is Θ-enabling, then $\mathcal{A}$ is Θ-enabling. □

Furthermore, all transitions of $\mathcal{A}_T^{\Theta_1}$ with a label from Θ_1 are by definition useful in $\mathcal{A}_T^{\Theta_1}$. Hence if there exists an $a \in \Sigma \cap \Theta_1$ which is enabled at every state of $\mathcal{A}_T^{\Theta_1}$, then all states of $\mathcal{A}_T^{\Theta_1}$ are reachable.

Lemma 3.2.48. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ, Θ_1 be two alphabets disjoint from Q and such that $\Theta \cap \Theta_1 \cap \Sigma \neq \varnothing$. Then*

if $\mathcal{A}_T^{\Theta_1}$ is Θ-enabling, then $Q = Q_{\mathcal{A},S}$.

Proof. Let $\mathcal{A}_T^{\Theta_1} = (Q, \Sigma, \delta_{\mathcal{A},T}^{\Theta_1}, I)$ be Θ-enabling. Since $Q_{\mathcal{A},S} \subseteq Q$ always holds, we only have to prove the converse inclusion $Q \subseteq Q_{\mathcal{A},S}$. Let $q \in Q$. Consider $a \in \Theta \cap \Theta_1 \cap \Sigma$. Then the assumption that $\mathcal{A}_T^{\Theta_1}$ is Θ-enabling implies there exists a $q' \in Q$ such that $(q, a, q') \in \delta_{\mathcal{A},T}^{\Theta_1}$. Since $a \in \Theta_1$, the definition of $\delta_{\mathcal{A},T}^{\Theta_1}$ implies that $(q, a, q') \in \delta_{\mathcal{A},T}$. Consequently, $q \in Q_{\mathcal{A},S}$. □

We have thus established that $\mathcal{A}$ is Θ-enabling whenever $\mathcal{A}_T^{\Theta_1}$ is. Conversely, $\mathcal{A}_T^{\Theta_1}$ obviously is Θ-enabling whenever $\mathcal{A}$ is and no action from Θ is included in both Θ_1 and the set of actions of $\mathcal{A}$. If the latter part of this condition is not met, then the Θ-enabling of $\mathcal{A}$ nevertheless does imply that $\mathcal{A}_T^{\Theta_1}$ is Θ-enabling if $\mathcal{A}$ is Θ_1-transition reduced.

Theorem 3.2.49. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ, Θ_1 be two alphabets disjoint from Q. Then*

> $\mathcal{A}_T^{\Theta_1}$ *is Θ-enabling if and only if $\mathcal{A}$ is Θ-enabling and $\mathcal{A} = \mathcal{A}_S = \mathcal{A}_T^{\Theta_1}$ whenever $\Theta \cap \Theta_1 \cap \Sigma \neq \varnothing$.*

Proof. (Only if) By Lemma 3.2.47, $\mathcal{A}$ is Θ-enabling if $\mathcal{A}_T^{\Theta_1}$ is Θ-enabling. Assume that $\Theta \cap \Theta_1 \cap \Sigma \neq \varnothing$. Then from Lemma 3.2.48 we know that the fact that $\mathcal{A}_T^{\Theta_1}$ is Θ-enabling implies that $Q = Q_{\mathcal{A},S}$. Consequently, $\delta = \delta \cap (Q_{\mathcal{A},S} \times \Sigma \times Q_{\mathcal{A},S})$ and so $\delta = \delta_{\mathcal{A},T}$. Thus we have $\mathcal{A} = \mathcal{A}_S$. Finally, by definition $\delta_{\mathcal{A},T} \subseteq \delta_{\mathcal{A},T}^{\Theta_1} \subseteq \delta$. Hence $\delta_{\mathcal{A},T} = \delta_{\mathcal{A},T}^{\Theta_1} = \delta$, which implies that $\mathcal{A} = \mathcal{A}_T^{\Theta_1}$.

(If) If $\mathcal{A}$ is Θ-enabling and $\mathcal{A} = \mathcal{A}_T^{\Theta_1}$, then it trivially follows that $\mathcal{A}_T^{\Theta_1}$ is Θ-enabling. Thus we assume that $\mathcal{A}$ is Θ-enabling and that $\Theta \cap \Theta_1 \cap \Sigma = \varnothing$. Let $\mathcal{A}_T^{\Theta_1} = (Q, \Sigma, \delta_{\mathcal{A},T}^{\Theta_1}, I)$. By definition $\delta_{\mathcal{A},T}^{\Theta_1} \supseteq \delta \setminus (Q \times \Theta_1 \times Q) = \delta \setminus (Q \times (\Theta_1 \cap \Sigma) \times Q)$. Since $\Theta \cap (\Theta_1 \cap \Sigma) = \varnothing$, it follows that $\delta_{\mathcal{A},T}^{\Theta_1} \supseteq \delta \cap (Q \times \Theta \times Q) = \delta \cap (Q \times (\Theta \cap \Sigma) \times Q)$. Consequently, we can apply Lemma 3.2.44 and conclude that $\mathcal{A}_T^{\Theta_1}$ is Θ-enabling. □

Corollary 3.2.50. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

> $\mathcal{A}_T^{\Theta}$ *is Θ-enabling if and only if $\mathcal{A}$ is Θ-enabling and $\mathcal{A} = \mathcal{A}_T^{\Theta}$.* □

Let us now focus on the interplay between active actions and enabled actions. Recall that whenever an action is active, then there exists at least one reachable state where it is enabled. Given an automaton we can thus delete the non-active actions from its alphabet and the transitions these actions are involved in from its transition relation, without effecting the enabling of this automaton.

Lemma 3.2.51. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ, Θ_1 be two alphabets disjoint from Q. Then*

> *if $\mathcal{A}$ is Θ-enabling, then $\mathcal{A}_A^{\Theta_1}$ is Θ-enabling.*

Proof. Let $\mathcal{A}$ be Θ-enabling. By definition $\mathcal{A}_A^{\Theta_1} = (Q, \Sigma_{\mathcal{A},A}^{\Theta_1}, \delta_{\mathcal{A},A}^{\Theta_1}, I)$, with $\Sigma_{\mathcal{A},A}^{\Theta_1} \subseteq \Sigma$ and $\delta_{\mathcal{A},A}^{\Theta_1} = \delta \cap (Q \times \Sigma_{\mathcal{A},A}^{\Theta_1} \times Q)$. Thus $\Theta \cap \Sigma_{\mathcal{A},A}^{\Theta_1} \subseteq \Theta \cap \Sigma$. Furthermore, $\delta_{\mathcal{A},A}^{\Theta_1} \supseteq \delta \cap (Q \times (\Theta \cap \Sigma_{\mathcal{A},A}^{\Theta_1}) \times Q)$. Consequently we can apply Lemma 3.2.44 and conclude that $\mathcal{A}_A^{\Theta_1}$ is Θ-enabling. □

The converse in general does not hold, even though $\mathcal{A}$ contains all transitions of $\mathcal{A}_A^{\Theta_1}$. The reason is that $\mathcal{A}$ may contain more actions than $\mathcal{A}_A^{\Theta_1}$ does. Thus whenever $\mathcal{A}_A^{\Theta_1}$ is Θ-enabling also $\mathcal{A}$ will be Θ-enabling, provided Θ contains no action of Θ_1 that is a non-active action of $\mathcal{A}$. Hence we require all actions from $\Theta_1 \cap \Theta$ that appear also in the set of actions of $\mathcal{A}$, to be active.

Lemma 3.2.52. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ, Θ_1 be two alphabets disjoint from Q and such that $\Theta \cap \Theta_1 \cap \Sigma \subseteq \Sigma_{\mathcal{A},A}$. Then*

if $\mathcal{A}_A^{\Theta_1}$ is Θ-enabling, then $\mathcal{A}$ is Θ-enabling.

Proof. Let $\mathcal{A}_A^{\Theta_1} = (Q, \Sigma_{\mathcal{A},A}^{\Theta_1}, \delta_{\mathcal{A},A}^{\Theta_1}, I)$ be Θ-enabling. By definition $\delta_{\mathcal{A},A}^{\Theta_1} \subseteq \delta$ and hence — once we have established that $\Theta \cap \Sigma \subseteq \Theta \cap \Sigma_{\mathcal{A},A}^{\Theta_1}$ — we can apply Lemma 3.2.44 and conclude that $\mathcal{A}$ is Θ-enabling.

Assume that $\Theta \cap \Theta_1 \cap \Sigma \subseteq \Sigma_{\mathcal{A},A}$. Now let $a \in \Theta \cap \Sigma$ and recall that $\Sigma_{\mathcal{A},A}^{\Theta_1} = (\Sigma \setminus \Theta_1) \cup (\Sigma_{\mathcal{A},A} \cap \Theta_1)$.
If $a \notin \Theta_1$, then $a \in (\Sigma \setminus \Theta_1) \subseteq \Sigma_{\mathcal{A},A}^{\Theta_1}$.
If $a \in \Theta_1$, then $a \in \Sigma_{\mathcal{A},A}$ by our assumption and thus $a \in \Sigma_{\mathcal{A},A}^{\Theta_1}$.

Hence in both cases $a \in \Theta \cap \Sigma_{\mathcal{A},A}^{\Theta_1}$ and we are done. □

From Lemma 3.2.2(3) we know that an action $a \in \Sigma$ of an automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$ is active if and only if there exists a useful transition $(q, a, q') \in \delta$. This means that $\Sigma_A = \varnothing$ whenever $Q_S = \varnothing$. If $Q_S \neq \varnothing$, however, and $\mathcal{A}$ is Θ-enabling, for some set of actions Θ, then every action in $\Theta \cap \Sigma$ is active in $\mathcal{A}$. This is due to the fact that a nonempty set of reachable states implies that all actions $\Theta \cap \Sigma$ are enabled in every initial state of $\mathcal{A}$, all of whose outgoing transitions are by definition useful.

Lemma 3.2.53. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton such that $Q_S \neq \varnothing$ and let Θ be an alphabet disjoint from Q. Then*

if $\mathcal{A}$ is Θ-enabling, then $\Theta \cap \Sigma \subseteq \Sigma_A$ and $\mathcal{A} = \mathcal{A}_A^{\Theta}$.

Proof. Let $\mathcal{A}$ be Θ-enabling and let $a \in \Theta \cap \Sigma$. Since $I = \varnothing$ implies that $Q_S = \varnothing$, it must be the case that $I \neq \varnothing$. Now let $q \in I$. Then there exists a $q' \in Q$ such that $(q, a, q') \in \delta$. Since $q \in I \subseteq Q_S$ is reachable in $\mathcal{A}$ this implies that a is active in $\mathcal{A}$, and thus $a \in \Sigma_A$. Hence $\Theta \cap \Sigma \subseteq \Sigma_A$.

Now let $\mathcal{A}_A^{\Theta} = (Q, \Sigma_{\mathcal{A},A}^{\Theta}, \delta_{\mathcal{A},A}^{\Theta}, I)$. Then $\Sigma_{\mathcal{A},A}^{\Theta} = (\Sigma \setminus \Theta) \cup (\Sigma_A \cap \Theta) = (\Sigma \setminus \Theta) \cup (\Sigma \cap \Theta) = \Sigma$ because $\Theta \cap \Sigma = \Theta \cap \Sigma_A$ by the above and $\Sigma_A \subseteq \Sigma$. By definition $\delta_{\mathcal{A},A}^{\Theta} = \delta \cap (Q \times \Sigma_{\mathcal{A},A}^{\Theta} \times Q)$. Hence $\delta_{\mathcal{A},A}^{\Theta} = \delta \cap (Q \times \Sigma \times Q) = \delta$. Consequently, $\mathcal{A}_A^{\Theta} = \mathcal{A}$. □

This lemma, together with Lemmata 3.2.51 and 3.2.52, directly implies the following theorem.

Theorem 3.2.54. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton such that $Q_S \neq \varnothing$ and let Θ, Θ_1 be two alphabets disjoint from Q. Then*

$$\mathcal{A} \textit{ is } \Theta\textit{-enabling if and only if } \mathcal{A}_A^{\Theta_1} \textit{ is } \Theta\textit{-enabling and } \Theta \cap \Theta_1 \cap \Sigma \subseteq \Sigma_{\mathcal{A},A}. \qquad \Box$$

Corollary 3.2.55. *Let $\mathcal{A}$ be an automaton and let Θ be an alphabet disjoint from its set of states. Then*

$$\mathcal{A} \textit{ is } \Theta\textit{-enabling if and only if } \mathcal{A}_A^{\Theta} \textit{ is } \Theta\textit{-enabling and } \mathcal{A} = \mathcal{A}_A^{\Theta}. \qquad \Box$$

In this subsection we have thus presented various conditions under which enabling is preserved from one (reduced) automaton to another. We have considered separately the state-reduced, action-reduced, and transition-reduced versions of automata. We now conclude with a result that incorporates also the reduced version of an automaton. It is obtained as a direct consequence of combining Theorem 3.2.46 with Corollary 3.2.55.

Theorem 3.2.56. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton. Then*

$$\textit{if } \mathcal{A} \textit{ is } \Sigma\textit{-enabling, then } \mathcal{A}_S = \mathcal{A}_R. \qquad \Box$$

3.2.3 Determinism

For an arbitrary automaton and a given action, it is in general not the case that for each of its states there is at most one possible way to execute this action. For certain types of systems (such as, e.g., transformational systems) it may however be crucial that the outcome of the execution of one of its actions is uniquely determined by the state the automaton is in. Thus when such a system is modeled as an automaton, the transition relation should contain at most one transition for each combination of such an action and a state of the automaton.

In a *deterministic* automaton, there is no choice as to what state the automaton ends up in after the execution of a sequence of actions. As was the case for enabling, the definition of determinism of an automaton is based on a local (state dependent) property of the transition relation.

Definition 3.2.57. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ be an alphabet disjoint from Q. Then*

> $\mathcal{A}$ *is* Θ-deterministic *if I contains at most one element and for all $a \in \Theta$ and for all $q \in Q$, $\{q' \in Q \mid (q, a, q') \in \delta\}$ contains at most one element.* $\Box$

Note the duality between enabling and determinism: given that a is an action of the automaton, then this automaton is $\{a\}$-enabling if each of its states has *at least* one outgoing a-transition, while it is $\{a\}$-deterministic if each of its states has *at most* one outgoing a-transition.

As in previous definitions, also the property of determinism is defined with respect to a separately specified arbitrary set of actions Θ. Similar to those previous notions, whether or not an automaton is Θ-deterministic is solely determined by those elements of Θ that are actions of $\mathcal{A}$. More precisely, if we assume that $\mathcal{A}$ contains at most one initial state, then $\mathcal{A}$ is always $\varnothing$-deterministic and — moreover — $\mathcal{A}$ is Θ-deterministic if and only if it is $\Theta \cap \Sigma$-deterministic, where Σ is the set of actions of $\mathcal{A}$.

Example 3.2.58. (Example 3.2.10 continued) Let $\mathcal{A}'$ be the automaton obtained from automaton $\mathcal{A}$ of Example 3.2.10 — depicted in Figure 3.3(a) — by replacing transition (q, a, q) with (q, b, q). Then $\mathcal{A}'$ is $\{a\}$-deterministic but not $\{b\}$-deterministic. Hence $\mathcal{A}'$ is neither $\{a, b\}$-deterministic. However, $\mathcal{A}'$ is $\{d\}$-deterministic, for all $d \notin \Sigma$, and thus $\{a, d\}$-deterministic as well. □

The deletion of states and/or transitions from an automaton does not affect its determinism of given actions.

Lemma 3.2.59. *Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, I_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, I_2)$ be two automata and let Θ_1, Θ_2 be two alphabets disjoint from $Q_1 \cup Q_2$. Let $\Theta_2 \cap \Sigma_2 \subseteq \Theta_1$, let $\delta_2 \cap (Q_2 \times \Theta_2 \times Q_2) \subseteq \delta_1$, and let I_2 contain at most one element. Then*

if $\mathcal{A}_1$ is Θ_1-deterministic, then $\mathcal{A}_2$ is Θ_2-deterministic.

Proof. Let $\mathcal{A}_1$ be Θ_1-deterministic. Now let $a \in \Theta_2$ and let $p \in Q_2$. Suppose that there exist $q, q' \in Q_2$ such that both $(p, a, q) \in \delta_2$ and $(p, a, q') \in \delta_2$. This implies that $a \in \Theta_2 \cap \Sigma_2$ and that both $(p, a, q) \in \delta_1$ and $(p, a, q') \in \delta_1$. Since $\Theta_2 \cap \Sigma_2 \subseteq \Theta_1$ and $\mathcal{A}_1$ is Θ_1-deterministic it follows that it must be the case that $q = q'$. Together with the fact that I_2 contains at most one element this implies that $\mathcal{A}_2$ is Θ_2-deterministic. □

This lemma has several immediate consequences.

Corollary 3.2.60. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ_1, Θ_2 be two alphabets disjoint from Q and such that $(\Theta_2 \cap \Sigma) \subseteq \Theta_1$. Then*

if $\mathcal{A}$ is Θ_1-deterministic, then $\mathcal{A}$ is Θ_2-deterministic. □

Corollary 3.2.61. *Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, I_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, I_2)$ be two automata such that $\mathcal{A}_2 \sqsubseteq \mathcal{A}_1$ and let Θ_1, Θ_2 be two alphabets disjoint from $Q_1 \cup Q_2$ and such that $(\Theta_2 \cap \Sigma_2) \subseteq \Theta_1$. Then* *if $\mathcal{A}_1$ is Θ_1-deterministic, then $\mathcal{A}_2$ is Θ_2-deterministic.* □

Corollary 3.2.62. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ and $\mathcal{A}' = (Q, \Sigma', \delta, I)$ be two automata such that $\Sigma \subseteq \Sigma'$ and let Θ be an alphabet disjoint from Q. Then*

if $\mathcal{A}$ is Θ-deterministic, then $\mathcal{A}'$ is Θ-deterministic. □

From the computational and the behavioral viewpoint also determinism is most relevant at the reachable states of an automaton. We thus finish this subsection with an overview of the influence that the determinism of one type of reduced automaton has on the determinism of another type of reduced automaton.

Theorem 3.2.63. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ, Θ_1 be two alphabets disjoint from Q. Then*

(1) if $\mathcal{A}$ is Θ-deterministic, then so is $\mathcal{A}_A^{\Theta_1}$,

(2) if $\mathcal{A}_A^{\Theta_1}$ is Θ-deterministic, then so is $\mathcal{A}_T^{\Theta_1}$, and

(3) if $\mathcal{A}_T^{\Theta_1}$ is Θ-deterministic, then so is $\mathcal{A}_S$.

Proof. (1) This follows directly from Corollary 3.2.61 since $\mathcal{A}_A^{\Theta_1}$ is a reduced version of $\mathcal{A}$ and thus $\mathcal{A}_A^{\Theta_1} \sqsubseteq \mathcal{A}$.

(2) Let $\mathcal{A}_A^{\Theta_1} = (Q, \Sigma_{\mathcal{A},A}^{\Theta_1}, \delta_{\mathcal{A},A}^{\Theta_1}, I)$ be Θ-deterministic. As by definition $\Sigma_{\mathcal{A},A}^{\Theta_1} \subseteq \Sigma$, Corollary 3.2.62 implies that also the automaton $\mathcal{A}' = (Q, \Sigma, \delta_{\mathcal{A},A}^{\Theta_1}, I)$ is Θ-deterministic. Now consider $\mathcal{A}_T^{\Theta_1} = (Q, \Sigma, \delta_{\mathcal{A},T}^{\Theta_1}, I)$. By definition $\delta_{\mathcal{A},T}^{\Theta_1} \subseteq \delta_{\mathcal{A},A}^{\Theta_1}$ and thus $\mathcal{A}_T^{\Theta_1} \sqsubseteq \mathcal{A}'$. Corollary 3.2.61 subsequently implies that also $\mathcal{A}_T^{\Theta_1}$ is Θ-deterministic.

(3) From Lemma 3.2.36(2) we know that $\mathcal{A}_S = (\mathcal{A}_T^{\Theta_1})_S$. Analogous to (1) the result now follows from the fact that $(\mathcal{A}_T^{\Theta_1})_S \sqsubseteq \mathcal{A}_T^{\Theta_1}$. □

In certain cases Θ-determinism is thus preserved from one automaton to another, for a set Θ of actions. The proof of this theorem however is heavily based on the containment of one automaton in another. In case the reverse of such a containment does not hold, often some characteristics crucial for preserving Θ-determinism from one automaton to another, are lacking. When formulating the reverses of the statements of this theorem, we thus settle for a demonstration of the preservation of determinism from one automaton to another for only a subset of Θ.

Theorem 3.2.64. *Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be an automaton and let Θ, Θ_1 be two alphabets disjoint from Q. Then*

(1) if $\mathcal{A}_S$ *is* Θ*-deterministic, then* $\mathcal{A}_T^{\Theta_1}$ *is* $(\Theta \cap \Theta_1)$*-deterministic,*

(2) if $\mathcal{A}_T^{\Theta_1}$ *is* Θ*-deterministic, then* $\mathcal{A}_A^{\Theta_1}$ *is* $(\Theta \setminus \Theta_1)$*-deterministic, and*

(3) if $\mathcal{A}_A^{\Theta_1}$ *is* Θ*-deterministic, then* $\mathcal{A}$ *is* $(\Theta \setminus (\Theta_1 \setminus \Sigma_{\mathcal{A},A}))$*-deterministic.*

Proof. (1) Let $\mathcal{A}_S = (Q_{\mathcal{A},S}, \Sigma_{\mathcal{A},A}, \delta_{\mathcal{A},T}, I)$ be Θ-deterministic. Now consider $\mathcal{A}_T^{\Theta_1} = (Q, \Sigma, \delta_{\mathcal{A},T}^{\Theta_1}, I)$. Since $(\Theta \cap \Theta_1) \cap \Sigma \subseteq \Theta$ and $\delta_{\mathcal{A},T}^{\Theta_1} \cap (Q \times (\Theta \cap \Theta_1) \times Q) \subseteq \delta_{\mathcal{A},T}$ it follows from Lemma 3.2.59 that $\mathcal{A}_T^{\Theta_1}$ is $(\Theta \cap \Theta_1)$-deterministic.

(2) Let $\mathcal{A}_T^{\Theta_1} = (Q, \Sigma, \delta_{\mathcal{A},T}^{\Theta_1}, I)$ be Θ-deterministic. Now consider $\mathcal{A}_A^{\Theta_1} = (Q, \Sigma_{\mathcal{A},A}^{\Theta_1}, \delta_{\mathcal{A},A}^{\Theta_1}, I)$. Since $(\Theta \setminus \Theta_1) \cap \Sigma_{\mathcal{A},A}^{\Theta_1} \subseteq \Theta$ and $\delta_{\mathcal{A},A}^{\Theta_1} \cap (Q \times (\Theta \setminus \Theta_1) \times Q) \subseteq \delta \cap (Q \times (\Sigma \setminus \Theta_1) \times Q) \subseteq \delta_{\mathcal{A},T}^{\Theta_1}$ it follows from Lemma 3.2.59 that $\mathcal{A}_A^{\Theta_1}$ is $(\Theta \setminus \Theta_1)$-deterministic.

(3) Let $\mathcal{A}_A^{\Theta_1} = (Q, \Sigma_{\mathcal{A},A}^{\Theta_1}, \delta_{\mathcal{A},A}^{\Theta_1}, I)$ be Θ-deterministic. Clearly $(\Theta \setminus (\Theta_1 \setminus \Sigma_{\mathcal{A},A})) \cap \Sigma \subseteq \Theta$. Moreover, since $\Theta \setminus (\Theta_1 \setminus \Sigma_{\mathcal{A},A})) = (\Theta \setminus \Theta_1) \cup (\Theta \cap (\Sigma_{\mathcal{A},A} \cap \Theta_1))$ it follows that $\delta \cap (Q \times (\Theta \setminus (\Theta_1 \setminus \Sigma_{\mathcal{A},A})) \times Q) \subseteq (\delta \cap (Q \times (\Sigma \setminus \Theta_1) \times Q)) \cup (\delta \cap (Q \times (\Sigma_{\mathcal{A},A} \cap \Theta_1) \times Q)) = \delta_{\mathcal{A},A}^{\Theta_1}$. Hence by Lemma 3.2.59 it follows that $\mathcal{A}$ is $(\Theta \setminus (\Theta_1 \setminus \Sigma_{\mathcal{A},A}))$-deterministic. $\square$

4. Synchronized Automata

In the previous chapter we have introduced automata as the basic components underlying team automata. In this chapter we define precisely how *automata* can be combined in order to form a *synchronized automaton*. Within such a synchronized automaton its constituting automata interact by synchronizing on certain occurrences of shared actions. We also define how to obtain a *subautomaton* from a synchronized automaton by focusing on a subset of its constituting automata, and we study the relation between synchronized automata and their subautomata in terms of computations. Consequently, we show how to iteratively obtain synchronized automata from synchronized automata.

We then characterize three basic and natural ways of synchronizing. We also define *maximal-syn synchronized automata* as the unique synchronized automata being maximal with respect to a given type of synchronization *syn*. Through the formulation of *predicates of synchronization* we furthermore provide direct descriptions of such synchronized automata. Finally, we conclude this chapter with a study of the effect that synchronizations have on the inheritance of the automata-theoretic properties introduced in Section 3.2 from synchronized automata to their (sub)automata, and vice versa.

Notation 1. *In this chapter we assume a fixed, but arbitrary and possibly infinite index set $\mathcal{I} \subseteq \mathbb{N}$, which we will use to index the automata involved. For each $i \in \mathcal{I}$, we let $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i, I_i)$ be a fixed automaton. Moreover, we let $\mathcal{S} = \{\mathcal{A}_i \mid i \in \mathcal{I}\}$ be a fixed set of automata. Note that $\mathcal{I} \subseteq \mathbb{N}$ implies that $\mathcal{I}$ is ordered by the usual $\leq$ relation on $\mathbb{N}$, thus inducing an ordering on $\mathcal{S}$. Also note that the $\mathcal{A}_i$ are not necessarily different.* $\square$

4.1 Definitions

We begin this section by defining synchronized automata as composite automata. Consequently, we consider also the dual approach by defining how to extract (sub)automata from a given synchronized automaton.

4.1.1 Synchronized Automata

Consider the set $\mathcal{S} = \{\mathcal{A}_i \mid i \in \mathcal{I}\}$ of automata, as fixed above. Then a state q of any synchronized automaton over $\mathcal{S}$ describes the states that each of the automata is in. The state space of any synchronized automaton $\mathcal{T}$ formed from $\mathcal{S}$ is thus the product $\prod_{i\in\mathcal{I}} Q_i$ of the state spaces of the automata of $\mathcal{S}$, with the product $\prod_{i\in\mathcal{I}} I_i$ of their initial states forming the set of initial states of $\mathcal{T}$.

The transition relation of such $\mathcal{T}$ is defined by allowing certain "synchronizations" and excluding others and is based solely on the transition relations of the automata forming the synchronized automaton.

Definition 4.1.1. *Let $a \in \bigcup_{i\in\mathcal{I}} \Sigma_i$. Then the* complete transition space *of a in $\mathcal{S}$ is denoted by $\Delta_a(\mathcal{S})$ and is defined as*

$$\Delta_a(\mathcal{S}) = \{(q, q') \in \textstyle\prod_{i\in\mathcal{I}} Q_i \times \prod_{i\in\mathcal{I}} Q_i \mid \exists j \in \mathcal{I} : \ \mathit{proj}_j^{[2]}(q, q') \in \delta_{j,a} \wedge \\ (\forall i \in \mathcal{I} : \ \mathit{proj}_i^{[2]}(q, q') \in \delta_{i,a} \vee \mathit{proj}_i(q) = \mathit{proj}_i(q'))\}. \ \square$$

The complete transition space $\Delta_a(\mathcal{S})$ thus consists of all possible combinations of a-transitions from automata of $\mathcal{S}$, with all non-participating automata remaining idle. It is an explicit requirement that at least one automaton is active, i.e. executes an a-transition. The transitions in $\Delta_a(\mathcal{S})$ are referred to as *synchronizations* (on a).

This $\Delta_a(\mathcal{S})$ is called the *complete* transition space of a in $\mathcal{S}$ because whenever a synchronized automaton $\mathcal{T}$ is constructed from $\mathcal{S}$, then for each action a, all a-transitions of $\mathcal{T}$ come from $\Delta_a(\mathcal{S})$. The transformation of a state of $\mathcal{T}$ is defined by the local state changes of the automata participating in the action of $\mathcal{T}$ being executed. When defining $\mathcal{T}$, for each action a, a specific subset δ_a of $\Delta_a(\mathcal{S})$ has to be chosen. By restricting the set of allowed transitions in this way, a certain kind of interaction between the automata constituting the synchronized automaton can be enforced.

Definition 4.1.2. *A* synchronized automaton over $\mathcal{S}$ *is a construct $\mathcal{T} = (Q, \Sigma, \delta, I)$, where*

$Q = \prod_{i\in\mathcal{I}} Q_i$,
$\Sigma = \bigcup_{i\in\mathcal{I}} \Sigma_i$,
$\delta \subseteq Q \times \Sigma \times Q$ *is such that for all $a \in \Sigma$,*

$$\delta_a \subseteq \Delta_a(\mathcal{S}), \text{ and}$$

$I = \prod_{i\in\mathcal{I}} I_i$. $\square$

All synchronized automata over a given set of automata thus have the same set of states, the same alphabet of actions, and the same set of initial states. They only differ by the choice of their transition relation, which is based on but not fixed by the transition relations of the individual automata. Due to this freedom of choosing a δ_a for each action a, a set of automata does not uniquely define a single synchronized automaton. Instead, a flexible framework is provided within which one can construct a variety of synchronized automata, all of which differ solely by the choice of the transition relation.

In the literature, automata are mostly composed according to some fixed strategy, thus leading to a uniquely defined synchronized automaton. In fact, the strategy that is prevalent in the literature (cf. the Introduction) is the rule to include, for all actions a, all and only those a-transitions in which all automata from $\mathcal{S}$ participate that have a as one of their actions. This leaves no choice for the transition relation and thus leads to a unique synchronized automaton. In Section 4.5 we will describe this and other fixed strategies for choosing transition relations in a predetermined way. Within our framework, however, it is precisely the freedom to choose transition relations which provides the flexibility to distinguish even the smallest nuances in the meaning of one's design.

The following example illustrates the definition of synchronized automata. Recall that vectors may be written vertically, even though in the text they are written horizontally.

Example 4.1.3. (Example 3.1.8 continued) Consider the automaton $W_2 = (\{s_2, t_2\}, \{a, b\}, \delta_2, \{s_2\})$, with $\delta_2 = \{(s_2, b, s_2), (s_2, a, t_2), (t_2, a, t_2), (t_2, b, s_2)\}$, modeling the second **w**heel of a car. Since W_2 in essence is just a copy of W_1 its structure is the same as that of W_1, depicted in Figure 3.1.

Now we show how W_1 and W_2 can form a synchronized automaton (an axle). The synchronized automaton $\mathcal{T}_{\{1,2\}}$ over $\{W_1, W_2\}$ is depicted in Figure 4.1(a). It has four states of which (s_1, s_2) is its only initial state. It has no other actions than a and b. We require the two wheels W_1 and W_2 to accelerate and break in unison, so we choose $\delta_{\{1,2\}} = \{((s_1, s_2), b, (s_1, s_2)), ((s_1, s_2), a, (t_1, t_2)), ((t_1, t_2), a, (t_1, t_2)), ((t_1, t_2), b, (s_1, s_2))\}$. We note that only the transition relation had to be chosen, all other elements follow from Definition 4.1.2.

Note that $\mathcal{T}_{\{1,2\}}$ is action reduced and transition reduced but not state reduced, since its states (s_1, t_2) and (t_1, s_2) are not reachable.

By choosing a different transition relation such as, e.g., $\delta'_{\{1,2\}} = \{((s_1, s_2), a, (s_1, t_2)), ((t_1, t_2), b, (s_1, s_2))\}$, another synchronized automaton over $\{W_1, W_2\}$ is defined, which we denote by $\mathcal{T}'_{\{1,2\}}$. Apart from its transition relation, $\mathcal{T}'_{\{1,2\}}$ contains the same elements as $\mathcal{T}_{\{1,2\}}$. $\mathcal{T}'_{\{1,2\}}$ is depicted in Figure 4.1(b).

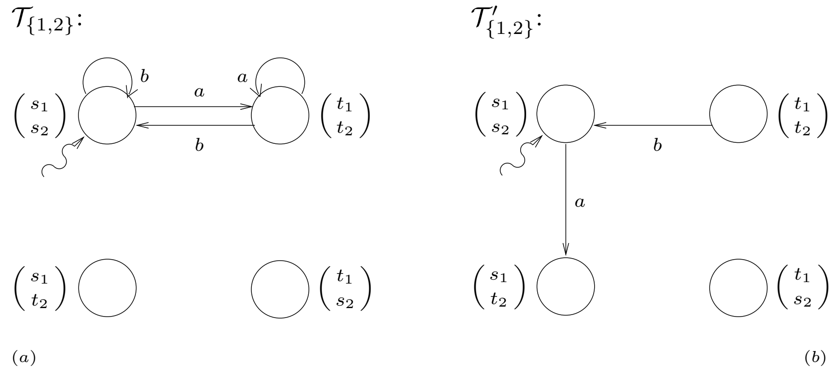


Fig. 4.1. Synchronized automata $\mathcal{T}_{\{1,2\}}$ and $\mathcal{T}'_{\{1,2\}}$.

If we assume that a flat tire is modeled by a wheel that cannot accelerate, then in $\mathcal{T}'_{\{1,2\}}$ the wheel W_1 has a flat tire. $\mathcal{T}'_{\{1,2\}}$ ends up in a deadlock (i.e. in a state where no action is enabled) after the execution of a, since one doesn't drive far with a flat tire. Furthermore, $\mathcal{T}'_{\{1,2\}}$ is not even action reduced nor is it transition reduced, because action b can never be executed in $\mathcal{T}'_{\{1,2\}}$ due to the fact that state (t_1, t_2) is not reachable. □

Definition 4.1.2 immediately implies the following result.

Theorem 4.1.4. *Every synchronized automaton is an automaton.* □

Since every synchronized automaton is again an automaton, it could in its turn be used as a constituting automaton of a new synchronized automaton.

Note, however, that even though a synchronized automaton over just one automaton $\{\mathcal{A}_j\}$ is again an automaton, such a synchronized automaton is different from its only constituting automaton. Even when Q_j and $\prod_{\{j\}} Q_j$ are identified, the transition relation of the synchronized automaton may be properly included in the transition relation of the automaton. This is due to the fact that the freedom in choosing the transition relation of a synchronized automaton, allows one to omit transitions from $\mathcal{A}_j$ in the transition relation of a synchronized automaton over $\{\mathcal{A}_j\}$.

Example 4.1.5. (Example 4.1.3 continued) We now show how to form a synchronized automaton (a car) over three automata (an axle and two wheels).

For $i \in \{3, 4\}$, let $W_i = (\{s_i, t_i\}, \{a, b\}, \delta_i, \{s_i\})$, where $\delta_i = \{(s_i, b, s_i), (s_i, a, t_i), (t_i, a, t_i), (t_i, b, s_i)\}$, be two automata modeling the third and the fourth **w**heel of a car. Since W_3 and W_4 (like W_2) are in essence just copies of W_1, their structure is the same as that of W_1, depicted in Figure 3.1.

Any synchronized automaton over $\{\mathcal{T}_{\{1,2\}}, W_3, W_4\}$ has alphabet $\{a, b\}$ and 16 states, among which the initial state $((s_1, s_2), s_3, s_4)$. We choose synchronized automaton $\hat{\mathcal{T}}$ by defining $\hat{\delta} = \{(((s_1, s_2), s_3, s_4), b, ((s_1, s_2), s_3, s_4)), (((s_1, s_2), s_3, s_4), a, ((t_1, t_2), t_3, t_4)), (((t_1, t_2), t_3, t_4), a, ((t_1, t_2), t_3, t_4)), (((t_1, t_2), t_3, t_4), b, ((s_1, s_2), s_3, s_4))\}$ as its transition relation. Its state-reduced version $\hat{\mathcal{T}}_S$ is depicted in Figure 4.2. □

$\hat{\mathcal{T}}_S$:

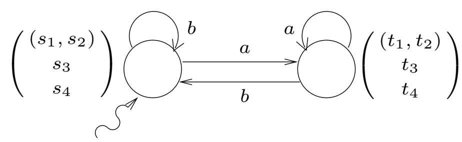


Fig. 4.2. State-reduced synchronized automaton $\hat{\mathcal{T}}_S$.

We conclude this section with two additional observations.

First it should be noted that in the definition of a synchronized automaton over $\mathcal{S} = \{\mathcal{A}_i \mid i \in \mathcal{I}\}$ we have implicitly used the ordering on $\mathcal{S}$ induced by $\mathcal{I}$. Every synchronized automaton over $\mathcal{S}$ has $\prod_{i \in \mathcal{I}} Q_i$ as its set of states and thus, if $\mathcal{I} = \{i_1, i_2, \ldots\}$ with $i_1 < i_2 < \cdots$, then every state q of $\mathcal{T}$ is a tuple $(q_1, q_2, \ldots)$ with $q_j \in Q_{i_j}$ for $j \geq 1$. This is convenient in concrete situations, but note that changing the order of the automata in $\mathcal{S}$ leads to formally different state spaces. As an example, consider two automata $\mathcal{A}_4$ and $\mathcal{A}_7$ with sets of states Q_4 and Q_7, respectively. Let $\mathcal{S} = \{\mathcal{A}_i \mid i \in \{4, 7\}\}$ and let $\mathcal{S}' = \{D_j \mid j \in \{1, 2\}\}$ with $D_1 = \mathcal{A}_7$ and $D_2 = \mathcal{A}_4$. Synchronized automata over $\mathcal{S}$ have $Q_4 \times Q_7$ as their state space, whereas synchronized automata over $\mathcal{S}'$ have $Q_7 \times Q_4$ as their state space. In Section 4.3 we will come back to the ordering within state spaces in a more general setup.

Secondly, neither in the definition of an automaton nor in the definition of a synchronized automaton, have we required a priori that states have to be reachable, that actions have to be active, or that transitions have to be useful in at least one computation starting from the initial state of the system. The lack of such extra conditions allows for a smooth and general definition of a synchronized automaton, with the full cartesian product of the sets of states of its constituting automata as the synchronized automaton's state space, the full union of the sets of actions of its constituting automata as its alphabet of actions, and an arbitrary selection of synchronizations as its transitions. Moreover, recall that in general no effective procedures exist to obtain the reduced versions of synchronized automata defined in Definitions 3.2.8, 3.2.9, and 3.2.27.

4.1.2 Subautomata

Given a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$, by focusing on a subset of the automata in $\mathcal{S}$, a subautomaton within $\mathcal{T}$ can be distinguished. Its transitions are restrictions of the transitions of $\mathcal{T}$ to the automata in the subset, while its actions of course are the actions of these automata.

Definition 4.1.6. *Let $\mathcal{T} = (Q, \Sigma, \delta, I)$ be a synchronized automaton over $\mathcal{S}$ and let $J \subseteq \mathcal{I}$. Then the* subautomaton of $\mathcal{T}$ determined by J *is denoted by $SUB_J(\mathcal{T})$ and is defined as $SUB_J(\mathcal{T}) = (Q_J, \Sigma_J, \delta_J, I_J)$, where*

$Q_J = \prod_{j \in J} Q_j$,
$\Sigma_J = \bigcup_{j \in J} \Sigma_j$,
$\delta_J \subseteq Q_J \times \Sigma_J \times Q_J$ *is such that for all $a \in \Sigma_J$,*

$$(\delta_J)_a = proj_J{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{A}_j \mid j \in J\}), \text{ and}$$

$I_J = \prod_{j \in J} I_j$. □

We write SUB_J instead of $SUB_J(\mathcal{T})$ if the synchronized automaton $\mathcal{T}$ is clear from the context. In Figure 4.3 we have sketched a subautomaton of a synchronized automaton.

The transition relation of a subautomaton SUB_J of a synchronized automaton $\mathcal{T}$ (over $\mathcal{S}$) determined by some $J \subseteq \mathcal{I}$, is obtained by restricting the transition relation of $\mathcal{T}$ to synchronizations between the automata in $\{\mathcal{A}_j \mid j \in J\}$. Hence in each transition of the subautomaton at least one of the automata from $\{\mathcal{A}_j \mid j \in J\}$ is actively involved. This is formalized by the intersection of $\mathrm{proj}_J{}^{[2]}(\delta_a)$ with $\Delta_a(\{\mathcal{A}_j \mid j \in J\})$, for each action a, as in each transition in this complete transition space at least one automaton from $\{\mathcal{A}_j \mid j \in \mathcal{J}\}$ is active.

Note that if $J = \varnothing$, then SUB_J is the trivial automaton.

Example 4.1.7. (Example 4.1.5 continued) Subautomaton $SUB_{\{1\}}(\mathcal{T}_{\{1,2\}}) = (\{(s_1), (t_1)\}, \{a, b\}, \delta_{\{1\}}, \{(s_1)\})$, where $\delta_{\{1\}} = \{((s_1), b, (s_1)), ((s_1), a, (t_1)), ((t_1), a, (t_1)), ((t_1), b, (s_1))\}$, is depicted in Figure 4.4(a).

Note that $SUB_{\{1\}}(\mathcal{T}_{\{1,2\}})$ differs from W_1 in the sense that it has (s_1) and (t_1) as states rather than s_1 and t_1. Obviously, $SUB_{\{1\}}(\mathcal{T}_{\{1,2\}})$ and W_1 do exhibit the same behavior.

$\mathcal{T}$ over $\mathcal{S} = \{\mathcal{A}_i \mid i \in \mathcal{I}\}$ with $\mathcal{I} = [n]$ for some even $n \geq 1$

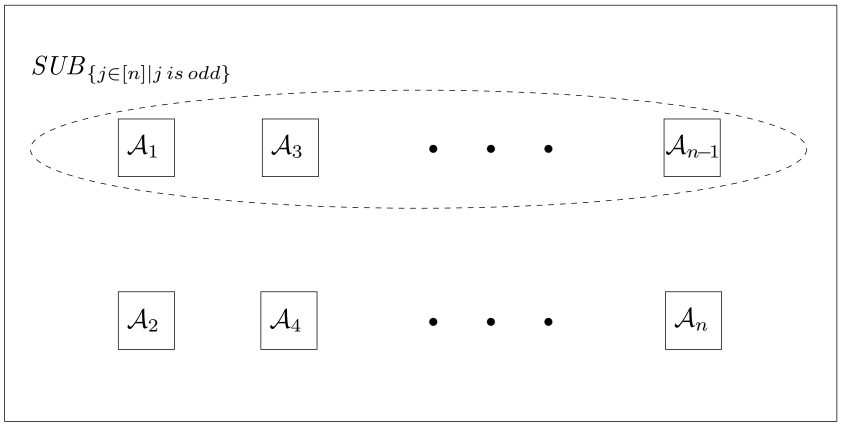


Fig. 4.3. Subautomaton $SUB_{\{j \in [n] \mid j \text{ is odd}\}}$ of synchronized automaton $\mathcal{T}$.

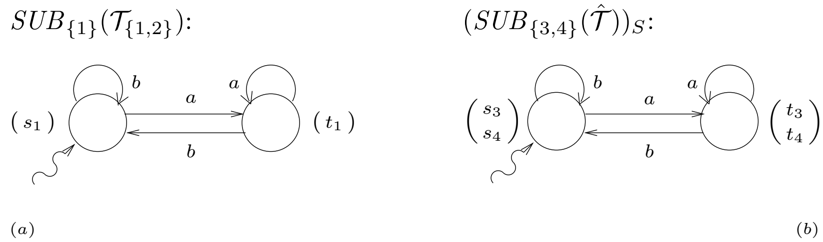


Fig. 4.4. Subautomaton $SUB_{\{1\}}(\mathcal{T}_{\{1,2\}})$ and automaton $(SUB_{\{3,4\}}(\hat{\mathcal{T}}))_S$.

Subautomaton $SUB_{\{3,4\}}(\hat{\mathcal{T}}) = (\{(s_3, s_4), (s_3, t_4), (t_3, s_4), (t_3, t_4)\}, \{a, b\}, \hat{\delta}_{\{3,4\}}, \{(s_3, s_4)\})$, where $\hat{\delta}_{\{3,4\}} = \{((s_3, s_4), b, (s_3, s_4)), ((s_3, s_4), a, (t_3, t_4)), ((t_3, t_4), a, (t_3, t_4)), ((t_3, t_4), b, (s_3, s_4))\}$, has as its state-reduced version the automaton $(SUB_{\{3,4\}}(\hat{\mathcal{T}}))_S$ depicted in Figure 4.4(b). $\square$

It is not hard to see that subautomata satisfy the requirements of a synchronized automaton.

Theorem 4.1.8. *Let* $\mathcal{T} = (Q, \Sigma, \delta, I)$ *be a synchronized automaton over* $\mathcal{S}$ *and let* $J \subseteq \mathcal{I}$. *Then*

SUB_J *is a synchronized automaton over* $\{\mathcal{A}_j \mid j \in J\}$.

Proof. The states, alphabet, and initial states of SUB_J as given in Definition 4.1.6 satisfy the requirements of Definition 4.1.2 for synchronized automata over $\{\mathcal{A}_j \mid j \in J\}$. Finally, $(\delta_J)_a \subseteq \Delta_a(\{\mathcal{A}_j \mid j \in J\})$ by Definition 4.1.6. $\square$

According to this theorem a subautomaton of a synchronized automaton is again a synchronized automaton and thus, by Theorem 4.1.4, also an automaton. In Section 4.3 we will consider the dual approach and use synchronized automata as automata in "larger" synchronized automata. It will be shown that subautomata can be used as automata to iteratively define the synchronized automaton they are derived from.

We conclude this section by comparing the set of transitions and computations of a singleton subautomaton $SUB_{\{j\}}$ of a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$ with those of the single automaton $\mathcal{A}_j$ from $\mathcal{S}$, where $j \in \mathcal{I}$. Due to the fact that $SUB_{\{j\}}$ has vectors (of one element) as states, whereas $\mathcal{A}_j$ does not, $SUB_{\{j\}}$ never equals $\mathcal{A}_j$ (see, e.g., Example 4.1.7). This is a purely syntactic reason, though. Therefore, in order to compare the set of transitions and computations of $\mathcal{A}_j$ with those of $SUB_{\{j\}}$, we identify $\prod_{\{j\}} Q_j$ and Q_j. To this end we define, for $j \in \mathcal{I}$, the homomorphism $\upsilon_j : (\Sigma \cup \prod_{\{j\}} Q_j)^\infty \to (\Sigma \cup Q_j)^\infty$ by

$$\upsilon_j(x) = \begin{cases} x & \text{if } x \in \Sigma \text{ and} \\ \mathrm{proj}_j(x) & \text{if } x \in \prod_{\{j\}} Q_j. \end{cases}$$

Consequently, we now show that for all $j \in \mathcal{I}$, the set of transitions (computations) of the subautomaton $SUB_{\{j\}}$ of a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$ is included in the set of transitions (computations) of the single automaton $\mathcal{A}_j$ from $\mathcal{S}$. However, as shown in the example directly following this result, these inclusions can be proper.

Lemma 4.1.9. *Let $\mathcal{T} = (Q, \Sigma, \delta, I)$ be a synchronized automaton over $\mathcal{S}$ and let $j \in \mathcal{I}$. Then*

(1) $\mathrm{proj}_j{}^{[2]}((\delta_{\{j\}})_a) \subseteq \delta_{j,a}$, for all $a \in \Sigma$, and

(2) $\upsilon_j(\mathbf{C}^\infty_{SUB_{\{j\}}}) \subseteq \mathbf{C}^\infty_{\mathcal{A}_j}$.

Proof. (1) Let $a \in \Sigma$ and let $(p, p') \in (\delta_{\{j\}})_a$. From Definition 4.1.6 then follows that $(p, p') \in \Delta_a(\{\mathcal{A}_j\}) = \{(q, q') \in \prod Q_j \times \prod Q_j \mid \mathrm{proj}_j{}^{[2]}(q, q') \in \delta_{j,a}\}$. Consequently, $\mathrm{proj}_j{}^{[2]}(p, p') \in \delta_{j,a}$.

(2) Let $\alpha \in \mathbf{C}^\infty_{SUB_{\{j\}}}$. First consider the finitary case, i.e. let $\alpha \in \mathbf{C}_{SUB_{\{j\}}}$. If $\alpha \in I_j$, then $\alpha = \prod_{\{j\}} q$ for some $q \in I_j$. Hence $\mathrm{proj}_j(\alpha) = q \in I_j$ and $\upsilon_j(\alpha) = q \in \mathbf{C}_{\mathcal{A}_j}$.

If $\alpha = \beta q a q'$ for some $\beta q \in \mathbf{C}_{SUB_{\{j\}}}$, $q, q' \in \prod_{\{j\}} Q_j$, and $a \in \Sigma_{\{j\}}$, with $(q, q') \in (\delta_{\{j\}})_a$, then we proceed with an inductive argument and assume that $\upsilon_j(\beta q) \in \mathbf{C}_{\mathcal{A}_j}$. From (1) follows that $\mathrm{proj}_j{}^{[2]}(q, q') \in \delta_{j,a}$ and we thus conclude $\upsilon_j(\alpha) = \upsilon_j(\beta)\mathrm{proj}_j(q) a \mathrm{proj}_j(q') \in \mathbf{C}_{\mathcal{A}_j}$.

Consequently consider the infinitary case, i.e. let $\alpha \in \mathbf{C}^{\omega}_{SUB_{\{j\}}}$. Let $\alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{SUB_{\{j\}}}$ be such that $\alpha = \lim_{n\to\infty} \alpha_n$. By the same reasoning as above $\upsilon_j(\alpha_n) \in \mathbf{C}_{\mathcal{A}_j}$, for all $n \geq 1$. Since υ_j is a letter-to-letter homomorphism we have $\upsilon_j(\alpha_1) \leq \upsilon_j(\alpha_2) \leq \cdots$ and $\lim_{n\to\infty} \upsilon_j(\alpha_n)$ is an infinite word. Furthermore $\lim_{n\to\infty} \upsilon_j(\alpha_n) = \upsilon_j(\lim_{n\to\infty} \alpha_n)$.

Hence $\upsilon_j(\alpha) = \upsilon_j(\lim_{n\to\infty} \alpha_n) = \lim_{n\to\infty} \upsilon_j(\alpha_n) \in \mathbf{C}^{\omega}_{\mathcal{A}_j}$. □

Given a synchronized automaton $\mathcal{T} = (Q, \Sigma, \delta, I)$ over $\mathcal{S}$, the following example shows that it can be the case that there exists a $j \in \mathcal{I}$ for which $\mathrm{proj}_j^{[2]}((\delta_{\{j\}})_a) \subset \delta_{j,a}$, for all $a \in \Sigma$, and $\upsilon_j(\mathbf{C}^{\infty}_{SUB_{\{j\}}}) \subset \mathbf{C}^{\infty}_{\mathcal{A}_j}$.

Example 4.1.10. Let $\mathcal{A}_1 = (\{q_1, q_1'\}, \{a\}, \{(q_1, a, q_1'), (q_1', a, q_1')\}, \{q_1\})$ and $\mathcal{A}_2 = (\{q_2, q_2'\}, \{a\}, \{(q_2, a, q_2')\}, \{q_2\})$ be the automata depicted in Figure 4.5(a).

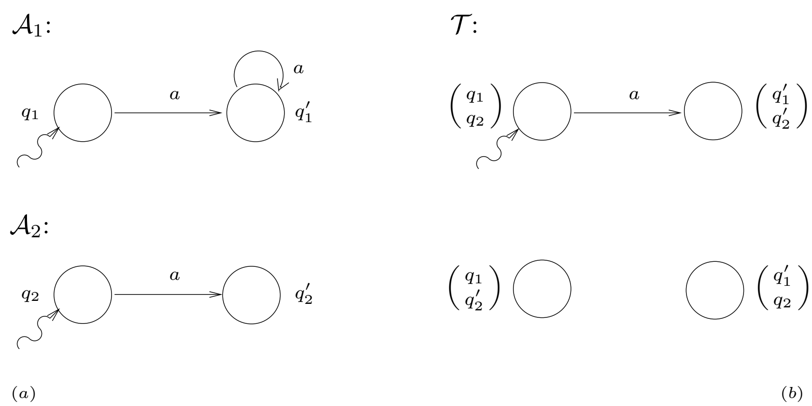


Fig. 4.5. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$, and synchronized automaton $\mathcal{T}$.

Consider the synchronized automaton $\mathcal{T} = (Q, \{a\}, \{((q_1, q_2), a, (q_1', q_2'))\}, \{(q_1, q_2)\})$, in which $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$, over $\{\mathcal{A}_1, \mathcal{A}_2\}$. It is depicted in Figure 4.5(b).

Let $j = 1$. It is clear that $(\delta_{\{1\}})_a = \{((q_1), (q_1'))\}$. Thus $\mathrm{proj}_1^{[2]}((\delta_{\{1\}})_a) = \{(q_1, q_1')\} \subset \{(q_1, q_1'), (q_1', q_1')\} = \delta_{1,a}$. Clearly, $\mathbf{C}_{SUB_{\{1\}}} = \{(q_1), (q_1)a(q_1')\}$. Hence $\upsilon_1(\mathbf{C}_{SUB_{\{1\}}}) = \{q_1, q_1 a q_1'\} \subset \{q_1, q_1 a q_1', q_1 a q_1' a q_1', \dots\} \cup \{q_1(a q_1')^{\omega}\} = \mathbf{C}^{\infty}_{\mathcal{A}_1}$. □

4.2 Projecting

In this section we want to extract the computations of any one of the (sub)automata constituting a synchronized automaton from the computations of this synchronized automaton. Note, however, that within the formalization of a synchronized automaton, no explicit information on loops is provided. That is to say, in general one cannot distinguish whether or not an automaton with a loop on a in its current local state participates in the synchronized automaton's synchronization on a. This automaton may have been idle or, after having participated in the action a starting from the global state, it may have returned to its original local state.

Example 4.2.1. Consider the three automata $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$, as depicted in Figure 4.6(a).

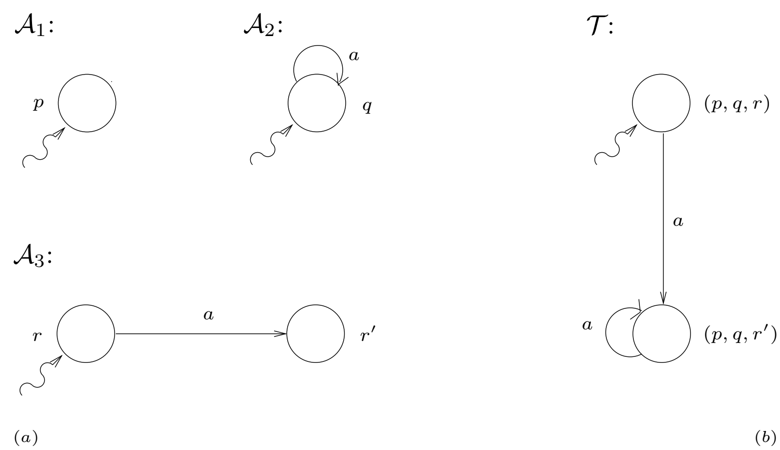


Fig. 4.6. Automata $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$, and synchronized automaton $\mathcal{T}$.

$\mathcal{A}_1$ and $\mathcal{A}_2$ each have only one state, p and q, respectively, which are their initial states. $\mathcal{A}_3$ has two states, r and r', of which r is its initial state. $\mathcal{A}_1$ has an empty alphabet, while both $\mathcal{A}_2$ and $\mathcal{A}_3$ have $\{a\}$ as their alphabet. Finally, $\mathcal{A}_1$ has no transitions at all, the transition relation of $\mathcal{A}_2$ consists solely of the loop (q, a, q), and that of $\mathcal{A}_3$ is $\{(r, a, r')\}$.

Now consider the synchronized automaton $\mathcal{T} = (\{(p, q, r), (p, q, r')\}, \{a\}, \delta, \{(p, q, r)\})$, where $\delta_a = \Delta_a(\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}) \setminus \{((p, q, r), a, (p, q, r))\}$, over $\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$. It is depicted in Figure 4.6(b). Now one might wonder which automata participate when the a-transitions of $\mathcal{T}$ are executed.

First consider the execution of the loop on a at (p, q, r') in $\mathcal{T}$. Clearly $\mathcal{A}_1$ does not participate as it cannot execute a at all. Also $\mathcal{A}_3$ does not participate since a is not enabled in r'. However, since in every transition of a synchronized automaton at least one component is required to participate, it must thus be the case that $\mathcal{A}_2$ executes its loop on a.

Secondly, consider the execution of the a-transition from (p, q, r) to (p, q, r') in $\mathcal{T}$. Clearly $\mathcal{A}_1$ is not involved. On the other hand, $\mathcal{A}_3$ is responsible for the local state change from r to r' and thus participates by executing a. But what about $\mathcal{A}_2$ — does it execute its loop on a or does it remain idle during this execution of a by $\mathcal{T}$? $\square$

In spite of the fact that Example 4.2.1 shows that information on the actual execution of loops by the constituting automata is lacking in the definition of a synchronized automaton, in order to relate the computations of a synchronized automaton to those taking place in its constituting automata we simply apply projections.

Recall that computations of a synchronized automaton are determined by the consecutive execution of transitions, starting from the initial state. Consider a transition (q, a, q') of a synchronized automaton over $\mathcal{S}$. We now assume that the j-th automaton participates in this transition by executing $(\mathrm{proj}_j(q), a, \mathrm{proj}_j(q'))$ whenever $\mathrm{proj}^{[2]}(q, q') \in \delta_{j,a}$. Otherwise no transition takes place in the j-th automaton. We thus resolve the lacking of information on loops by assuming that the presence of an automaton's loop in a transition of a synchronized automaton implies execution of that loop. This may be considered as a "maximal" interpretation of the participation of its constituting automata in transitions of synchronized automata, in the sense that we assume that if an automaton could have participated in an a-transition of the synchronized automaton by executing a loop on this action a, then it indeed has done so.

Example 4.2.2. (Example 4.2.1 continued) We consider the abovementioned maximal interpretation of the automata's participation in transitions of the synchronized automaton. Then $\mathcal{A}_2$ is thus assumed to execute its loop on a at q during the execution of a at (p, q, r) by means of the a-transition $((p, q, r), (p, q, r'))$ of $\mathcal{T}$. $\square$

Using the maximal interpretation we define the projection on (sub)automata of the computations of a synchronized automaton. Because of the results at the end of Section 4.1 we define separately the projection on the subautomaton defined by $\{j\}$ of a synchronized automaton and the projection on its j-th automaton. The formal reason behind this is the fact that Q_j and $\prod_{\{j\}} Q_j$

are not identified. In fact, as we will show shortly, the two separate definitions are the same whenever Q_j and $\prod_{\{j\}} Q_j$ are identified.

Finally, one could think of other interpretations of the participation of constituting (sub)automata in transitions of synchronized automata in case of loops.

Definition 4.2.3. *Let $\mathcal{T} = (Q, \Sigma, \delta, I)$ be a synchronized automaton over $\mathcal{S}$. Let $J \subseteq \mathcal{I}$. Then*

(1) the projection on subautomaton SUB_J of a finite computation $\alpha \in \mathbf{C}_{\mathcal{T}}$ *is denoted by $\pi_{SUB_J}(\alpha)$ and is defined as*
   (a) if $\alpha = q \in I$, then $\pi_{SUB_J}(\alpha) = proj_J(q)$, and
   (b) if $\alpha = \beta q a q'$, for some $\beta q \in \mathbf{C}_{\mathcal{T}}$, $q, q' \in Q$, and $a \in \Sigma$, then

$$\pi_{SUB_J}(\alpha) = \begin{cases} \pi_{SUB_J}(\beta q) & \text{if } proj_J{}^{[2]}(q, q') \notin (\delta_J)_a \text{ and} \\ \pi_{SUB_J}(\beta q) a \, proj_J(q') & \text{if } proj_J{}^{[2]}(q, q') \in (\delta_J)_a, \end{cases}$$

   and

(2) the projection on subautomaton SUB_J of an infinite computation $\alpha \in \mathbf{C}_{\mathcal{T}}^{\omega}$ *is denoted by $\pi_{SUB_J}(\alpha)$ and is defined as*

$$\pi_{SUB_J}(\alpha) = \lim_{n\to\infty} \pi_{SUB_J}(\alpha_n) \text{ whenever } \alpha = \lim_{n\to\infty} \alpha_n \text{ for } \alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{\mathcal{T}}.$$

Let $j \in \mathcal{I}$. Then

(3) the projection on automaton $\mathcal{A}_j$ of a finite computation $\alpha \in \mathbf{C}_{\mathcal{T}}$ *is denoted by $\pi_{\mathcal{A}_j}(\alpha)$ and is defined as*
   (a) if $\alpha = q \in I$, then $\pi_{\mathcal{A}_j}(\alpha) = proj_j(q)$, and
   (b) if $\alpha = \beta q a q'$, for some $\beta q \in \mathbf{C}_{\mathcal{T}}$, $q, q' \in Q$, and $a \in \Sigma$, then

$$\pi_{\mathcal{A}_j}(\alpha) = \begin{cases} \pi_{\mathcal{A}_j}(\beta q) & \text{if } proj_j{}^{[2]}(q, q') \notin \delta_{j,a} \text{ and} \\ \pi_{\mathcal{A}_j}(\beta q) a \, proj_j(q') & \text{if } proj_j{}^{[2]}(q, q') \in \delta_{j,a}, \end{cases}$$

   and

(4) the projection on automaton $\mathcal{A}_j$ of an infinite computation $\alpha \in \mathbf{C}_{\mathcal{T}}^{\omega}$ *is denoted by $\pi_{\mathcal{A}_j}(\alpha)$ and is defined as*

$$\pi_{\mathcal{A}_j}(\alpha) = \lim_{n\to\infty} \pi_{\mathcal{A}_j}(\alpha_n) \text{ whenever } \alpha = \lim_{n\to\infty} \alpha_n \text{ for } \alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{\mathcal{T}}. \ \Box$$

Recall that every prefix of odd length of an infinite computation α of a synchronized automaton $\mathcal{T}$ is a finite computation. Thus α is the limit of any prefix-ordered infinite subset of its finite prefixes. Moreover, if $\alpha_1 \leq \alpha_2$ for finite computations α_1 and α_2 of $\mathcal{T}$, then $\pi_{SUB_J}(\alpha_1) \leq \pi_{SUB_J}(\alpha_2)$, for all

$J \subseteq \mathcal{I}$, and $\pi_{\mathcal{A}_j}(\alpha_1) \leq \pi_{\mathcal{A}_j}(\alpha_2)$, for all $j \in \mathcal{I}$. Hence the projection $\pi_{SUB_J}(\alpha)$ on subautomaton $SUB_J(\mathcal{T})$ and the projection $\pi_{\mathcal{A}_j}(\alpha)$ on automaton $\mathcal{A}_j$ are well defined for any computation $\alpha \in \mathbf{C}^\infty_\mathcal{T}$. Furthermore, $\pi_{SUB_J}(\lim_{n\to\infty} \alpha_n) = \lim_{n\to\infty} \pi_{SUB_J}(\alpha_n)$ and $\pi_{\mathcal{A}_j}(\lim_{n\to\infty} \alpha_n) = \lim_{n\to\infty} \pi_{\mathcal{A}_j}(\alpha_n)$.

Note that $\pi_{SUB_J}(\alpha)$ and $\pi_{\mathcal{A}_j}(\alpha)$ can be finite sequences. This happens if subautomaton $SUB_J(\mathcal{T})$ or automaton $\mathcal{A}_j$, respectively, no longer participates in α after a finite number k of steps. In that case, if $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots$, then $\pi_{SUB_J}(q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n) = \pi_{SUB_J}(q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n a_{n+1} q_{n+1})$, for all $n \geq k$, and hence $\pi_{SUB_J}(\alpha) = \pi_{SUB_J}(q_0 a_1 q_1 a_2 q_2 \cdots a_k q_k)$. Likewise $\pi_{\mathcal{A}_j}(\alpha) = \pi_{\mathcal{A}_j}(q_0 a_1 q_1 a_2 q_2 \cdots a_k q_k)$ in that case.

Contrary to what one might expect from Example 4.1.10, we indeed see that for each computation of a synchronized automaton its projection on an automaton "agrees" with its projection on the corresponding singleton subautomaton, in the sense that they are equal whenever Q_j and $\prod_{\{j\}} Q_j$ are identified.

Theorem 4.2.4. *Let $\mathcal{T} = (Q, \Sigma, \delta, I)$ be a synchronized automaton over $\mathcal{S}$ and let $j \in \mathcal{I}$. Then*

$$\upsilon_j(\pi_{SUB_{\{j\}}}(\mathbf{C}^\infty_\mathcal{T})) = \pi_{\mathcal{A}_j}(\mathbf{C}^\infty_\mathcal{T}).$$

Proof. Let $\alpha \in \mathbf{C}^\infty_\mathcal{T}$. First consider the finitary case, i.e. let $\alpha \in \mathbf{C}_\mathcal{T}$. We proceed by induction on the length of w. If $\alpha = q$, then $\alpha \in \prod_{i\in\mathcal{I}} I_i$. By Definition 4.2.3, $\pi_{\mathcal{A}_j}(\alpha) = \mathrm{proj}_j(\alpha)$ and $\pi_{SUB_{\{j\}}}(\alpha) = \mathrm{proj}_{\{j\}}(\alpha)$. Consequently $\upsilon_j(\pi_{SUB_{\{j\}}}(\alpha)) = \mathrm{proj}_j(\mathrm{proj}_{\{j\}}(\alpha)) = \mathrm{proj}_j(\alpha) = \pi_{\mathcal{A}_j}(\alpha)$.

Next assume that $\alpha = \beta q a q'$ for some $\beta \in (\Sigma \cup Q)^*$, $q, q' \in Q$, and $a \in \Sigma$, such that $\beta q \in \mathbf{C}_\mathcal{T}$ and $(q, q') \in \delta_a$. It is not difficult to see that ${\mathrm{proj}_j}^{[2]}(q, q') \in \delta_{j,a}$ if and only if ${\mathrm{proj}_{\{j\}}}^{[2]}(q, q') \in (\delta_{\{j\}})_a$. Indeed we already know from Lemma 4.1.9 that ${\mathrm{proj}_{\{j\}}}^{[2]}((\delta_{\{j\}})_a) \subseteq \delta_{j,a}$ and hence ${\mathrm{proj}_{\{j\}}}^{[2]}(q, q') \in (\delta_{\{j\}})_a$ implies ${\mathrm{proj}_j}^{[2]}({\mathrm{proj}_{\{j\}}}^{[2]}(q, q')) = {\mathrm{proj}_j}^{[2]}(q, q') \in \delta_{j,a}$. Conversely, if ${\mathrm{proj}_j}^{[2]}(q, q') \in \delta_{j,a}$ then ${\mathrm{proj}_{\{j\}}}^{[2]}(q, q') \in (\delta_{\{j\}})_a$ provided that $(q, q') \in \delta_a$, which is the case. Returning to our computation α we now obtain the following.

If ${\mathrm{proj}_j}^{[2]}(q, q') \notin \delta_{j,a}$, then by induction $\pi_{\mathcal{A}_j}(\alpha) = \pi_{\mathcal{A}_j}(\beta q)$ and $\pi_{\mathcal{A}_j}(\beta q) = \upsilon_j(\pi_{SUB_{\{j\}}}(\beta q))$. As ${\mathrm{proj}_{\{j\}}}^{[2]}(q, q') \notin (\delta_{\{j\}})_a$ it follows that $\pi_{SUB_{\{j\}}}(\alpha) = \pi_{SUB_{\{j\}}}(\beta q)$. Consequently $\pi_{\mathcal{A}_j}(\alpha) = \upsilon_j(\pi_{SUB_{\{j\}}}(\alpha))$.

If ${\mathrm{proj}_j}^{[2]}(q, q') \in \delta_{j,a}$, then by induction $\pi_{\mathcal{A}_j}(\alpha) = \pi_{\mathcal{A}_j}(\beta q) a \mathrm{proj}_j(q') = \upsilon_j(\pi_{SUB_{\{j\}}}(\beta q)) a \mathrm{proj}_j(q')$. As ${\mathrm{proj}_{\{j\}}}^{[2]}(q, q') \in (\delta_{\{j\}})_a$, then $\pi_{SUB_{\{j\}}}(\alpha) = \pi_{SUB_{\{j\}}}(\beta q) a \mathrm{proj}_{\{j\}}(q')$. Hence $\pi_{\mathcal{A}_j}(\alpha) = \upsilon_j(\pi_{SUB_{\{j\}}}(\beta q) a \mathrm{proj}_{\{j\}}(q')) = \upsilon_j(\pi_{SUB_{\{j\}}}(\alpha))$. This concludes the proof for the finitary case.

Now consider the infinitary case, i.e. let $\alpha \in \mathbf{C}_{\mathcal{T}}^{\omega}$. Let $\alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{\mathcal{T}}$ be such that $\alpha = \lim_{n\to\infty} \alpha_n$. Then by definition $\pi_{\mathcal{A}_j}(\alpha) = \lim_{n\to\infty} \pi_{\mathcal{A}_j}(\alpha_n)$ and $\pi_{SUB_{\{j\}}}(\alpha) = \lim_{n\to\infty} \pi_{SUB_{\{j\}}}(\alpha_n)$. By the same reasoning as above $\pi_{\mathcal{A}_j}(\alpha_n) = \upsilon_j(\pi_{SUB_{\{j\}}}(\alpha_n))$ and since υ_j is a homomorphism we thus obtain $\pi_{\mathcal{A}_j}(\alpha) = \lim_{n\to\infty} \upsilon_j(\pi_{SUB_{\{j\}}}(\alpha_n)) = \upsilon_j(\lim_{n\to\infty} \pi_{SUB_{\{j\}}}(\alpha_n)) = \upsilon_j(\pi_{SUB_{\{j\}}}(\alpha))$. □

Example 4.2.5. (Example 4.1.10 continued) It is easy to see that $\mathbf{C}_{\mathcal{T}} = \{(q_1, q_2), (q_1, q_2)a(q_1', q_2')\}$. Now recall that $j = 1$. Then $\upsilon_1(\pi_{SUB_{\{1\}}}(\mathbf{C}_{\mathcal{T}})) = \upsilon_1(\{(q_1), (q_1)a(q_1')\}) = \{q_1, q_1 a q_1'\} = \pi_{\mathcal{A}_1}(\mathbf{C}_{\mathcal{T}})$. □

We conclude this section by showing that if we take the set of computations of a synchronized automaton and consequently project on a (sub)automaton of that synchronized automaton, then the result is always included in the set of computations of that (sub)automaton. However, these inclusions may be proper.

Lemma 4.2.6. *Let $\mathcal{T} = (Q, \Sigma, \delta, I)$ be a synchronized automaton over $\mathcal{S}$ and let $J \subseteq \mathcal{I}$. Then*

$$\pi_{SUB_J}(\mathbf{C}_{\mathcal{T}}^{\infty}) \subseteq \mathbf{C}_{SUB_J}^{\infty}.$$

Proof. Let $\alpha \in \mathbf{C}_{\mathcal{T}}^{\infty}$. First consider the finitary case, i.e. let $\alpha \in \mathbf{C}_{\mathcal{T}}$. Hence $\alpha = q_0 a_1 q_1 a_2 \cdots a_n q_n$ for some $n \geq 0$, $q_\ell \in Q$ for $0 \leq \ell \leq n$, and $a_\ell \in \Sigma$ for $1 \leq \ell \leq n$. By Definition 4.2.3 we have $\pi_{SUB_J}(\alpha) = p_0 b_1 p_1 b_2 \cdots b_m p_m$ for some $m \geq 0$, $p_\ell \in Q_J$ for $0 \leq \ell \leq m$, and $b_\ell \in \Sigma_J$ for $1 \leq \ell \leq m$.
We prove by induction on n that $\pi_{SUB_J}(\alpha) \in \mathbf{C}_{SUB_J}$ and, furthermore, that $\mathrm{proj}_J(q_n) = p_m$.

If $n = 0$, then $\alpha = q_0 \in I$. Thus by Definition 4.2.3 we have $\pi_{SUB_J}(\alpha) = \mathrm{proj}_J(q_0) \in I_J$, which implies that $\pi_{SUB_J}(\alpha) \in \mathbf{C}_{SUB_J}$. Moreover, $m = 0$ and $\mathrm{proj}_J(q_0) = p_0$.

Now assume that the statement holds for some $k \geq 0$. Let $n = k + 1$. Then by Definition 4.2.3 we have $\pi_{SUB_J}(\alpha) = \pi_{SUB_J}(q_0 a_1 q_1 a_2 \cdots a_k q_k)\gamma$, where $\gamma = \lambda$ if $\mathrm{proj}_J^{[2]}(q_k, q_{k+1}) \notin (\delta_J)_{a_{k+1}}$ and $\gamma = a_{k+1}\mathrm{proj}_J(q_{k+1})$ otherwise.
First consider the case $\gamma = \lambda$. Then $\pi_{SUB_J}(\alpha) \in \mathbf{C}_{SUB_J}$ by the induction hypothesis. Moreover, since $\mathrm{proj}_J^{[2]}(q_k, q_{k+1}) \notin (\delta_J)_{a_{k+1}}$, Definition 4.1.1 implies that $\mathrm{proj}_J(q_k) = \mathrm{proj}_J(q_{k+1})$. By the induction hypothesis $\mathrm{proj}_J(q_k) = p_m$, and hence $\mathrm{proj}_J(q_{k+1}) = p_m$.
Secondly, consider the case $\gamma \neq \lambda$. Then $\pi_{SUB_J}(\alpha) = p_0 b_1 p_1 b_2 \cdots b_m p_m = \pi_{SUB_J}(q_0 a_1 q_1 a_2 \cdots a_k q_k) a_{k+1} \mathrm{proj}_J(q_{k+1})$. Thus in this case $b_m = a_{k+1}$ and $p_m = \mathrm{proj}_J(q_{k+1})$.

The only thing left to prove is that $\pi_{SUB_J}(\alpha) \in \mathbf{C}_{SUB_J}$. We already have that $\mathrm{proj}_J{}^{[2]}(q_k, q_{k+1}) \in (\delta_J)_{a_{k+1}}$. From the induction hypothesis above it now follows that $p_0 b_1 p_1 b_2 \cdots b_{m-1} p_{m-1} \in \mathbf{C}_{SUB_J}$ and $p_{m-1} = \mathrm{proj}_J(q_k)$. Thus $\mathrm{proj}_J{}^{[2]}(p_{m-1}, p_m) = \mathrm{proj}_J{}^{[2]}(q_k, q_{k+1}) \in (\delta_J)_{b_m}$, which implies $\pi_{SUB_J}(\alpha) = p_0 b_1 p_1 b_2 \cdots b_m p_m \in \mathbf{C}_{SUB_J}$.

Now consider the infinitary case, i.e. let $\alpha \in \mathbf{C}^{\omega}_{\mathcal{T}}$. Hence $\alpha = \lim_{n\to\infty} \alpha_n$ for finite computations $\alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{\mathcal{T}}$. Then $\pi_{SUB_J}(\alpha_1) \leq \pi_{SUB_J}(\alpha_2) \leq \cdots$ and $\pi_{SUB_J}(\alpha_n) \in \mathbf{C}_{SUB_J}$, for all $n \geq 1$. Thus $\pi_{SUB_J}(\alpha) = \lim_{n\to\infty} \pi_{SUB_J}(\alpha_n) \in \mathbf{C}^{\infty}_{SUB_J}$. □

Corollary 4.2.7. *Let $\mathcal{T}$ be a synchronized automaton over $\mathcal{S}$ and let $j \in \mathcal{I}$. Then*

$$\pi_{\mathcal{A}_j}(\mathbf{C}^{\infty}_{\mathcal{T}}) \subseteq \mathbf{C}^{\infty}_{\mathcal{A}_j}.$$

Proof. Directly from Theorem 4.2.4 and Lemmata 4.2.6 and 4.1.9. □

In the following example we show that, given a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$, it can be the case that there exists a subset $J \subseteq \mathcal{I}$ or a $j \in \mathcal{I}$ for which $\pi_{SUB_J}(\mathbf{C}^{\infty}_{\mathcal{T}}) \subset \mathbf{C}^{\infty}_{SUB_J}$ or $\pi_{\mathcal{A}_j}(\mathbf{C}^{\infty}_{\mathcal{T}}) \subset \mathbf{C}^{\infty}_{\mathcal{A}_j}$, respectively.

Example 4.2.8. Let $\mathcal{A}_1 = (\{q_1, q_1'\}, \{a, b\}, \{(q_1, a, q_1), (q_1, b, q_1')\}, \{q_1\})$ and $\mathcal{A}_2 = (\{q_2, q_2'\}, \{a\}, \{(q_2, a, q_2')\}, \{q_2\})$ be the automata depicted in Figure 4.7(a).

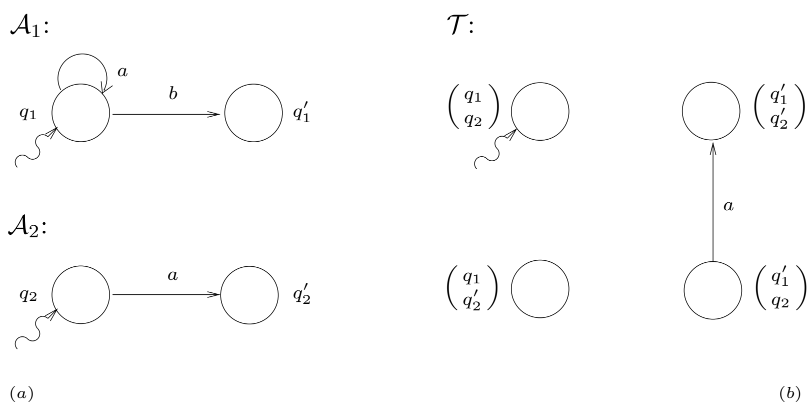


Fig. 4.7. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$, and synchronized automaton $\mathcal{T}$.

Consider synchronized automaton $\mathcal{T} = (Q, \{a, b\}, \{((q_1', q_2), a, (q_1', q_2'))\}, \{(q_1, q_2)\})$, in which $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$, over $\{\mathcal{A}_1, \mathcal{A}_2\}$. It is depicted in Figure 4.7(b).

It is clear that (q_1, q_2) is the only computation of $\mathcal{T}$, whereas $SUB_{\{2\}}$ has the two computations (q_2) and $(q_2)a(q_2')$. Hence we have $\pi_{SUB_{\{2\}}}(\mathbf{C}^{\infty}_{\mathcal{T}}) = \mathrm{proj}_{\{2\}}((q_1, q_2)) = (q_2) \subset \{(q_2), (q_2)a(q_2')\} = \mathbf{C}^{\infty}_{SUB_{\{2\}}}$ and, according to Lemma 4.1.9(2) and Theorem 4.2.4, $\pi_{\mathcal{A}_2}(\mathbf{C}^{\infty}_{\mathcal{T}}) = \upsilon_2(\pi_{SUB_{\{2\}}}(\mathbf{C}^{\infty}_{\mathcal{T}})) = \upsilon_2((q_2)) = q_2 \subset \{q_2, q_2 a q_2'\} = \upsilon_2(\{(q_2), (q_2)a(q_2')\}) = \upsilon_2(\mathbf{C}^{\infty}_{SUB_{\{2\}}}) \subseteq \mathbf{C}^{\infty}_{\mathcal{A}_2}$.

As a further example we consider the synchronized automaton $\mathcal{T}' = (Q, \{a, b\}, \{((q_1, q_2), a, (q_1, q_2'))\}, \{(q_1, q_2)\})$ over $\{\mathcal{A}_1, \mathcal{A}_2\}$. It is depicted in Figure 4.8.

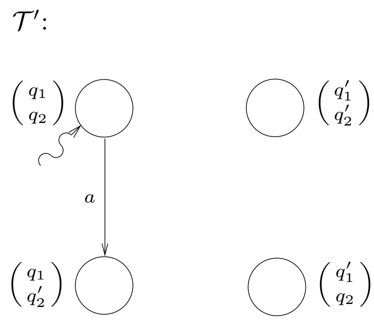


Fig. 4.8. Synchronized automaton $\mathcal{T}'$.

It is clear that $\mathbf{C}^{\infty}_{\mathcal{T}'} = \{(q_1, q_2), (q_1, q_2)a(q_1, q_2')\}$, whereas we have $\mathbf{C}^{\infty}_{\mathcal{A}_1} = \{q_1, q_1 a q_1, q_1 b q_1', q_1 a q_1 a q_1, q_1 a q_1 b q_1', \dots\} \cup \{q_1(a q_1)^{\omega}\}$. Hence we now see that $\pi_{\mathcal{A}_1}(\mathbf{C}^{\infty}_{\mathcal{T}'}) = \{q_1, q_1 a q_1\} \subset \mathbf{C}^{\infty}_{\mathcal{A}_1}$. $\square$

4.3 Iterated Composition

In this section we show that synchronized automata are naturally suited to describe hierarchical systems. We do this by demonstrating how to iteratively build synchronized automata from synchronized automata, and how to consider subautomata as constituting automata in an iterated definition of a synchronized automaton.

Given a set of automata $\mathcal{S}$, there may be several ways of forming a synchronized automaton over $\mathcal{S}$. Until now we directly defined synchronized automata over $\mathcal{S}$, but other routes are also feasible. We might first (iteratively) form synchronized automata from (disjoint) subsets of $\mathcal{S}$ and then use these as automata for a higher-level synchronized automaton, until after a finite number of such iterations all automata from $\mathcal{S}$ have been used. This is shown in Example 4.1.5 and Figure 4.2, where four wheels are combined by first connecting two of them (to form an axle) and then attaching the other two to the result. This section shows that whatever route chosen, the resulting *iterated* synchronized automaton can always be regarded as a synchronized automaton over $\mathcal{S}$: it will always have the same alphabet of actions and it will have essentially the same state space, transition space, and set of initial states as any synchronized automaton formed directly over $\mathcal{S}$.

Example 4.3.1. Let $\mathcal{S} = \{A_i \mid i \in [7]\}$, with $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i, I_i)$, for $i \in [7]$. Let $\mathcal{T}_{1-7} = (\prod_{i\in[7]} Q_i, \bigcup_{i\in[7]} \Sigma_i, \delta, \prod_{i\in[7]} I_i)$ be a synchronized automaton over $\mathcal{S}$. As δ is not relevant for the moment, it is not specified any further. Recall that all other parameters of $\mathcal{T}_{1-7}$ are uniquely defined by Definition 4.1.2. The structure of this synchronized automaton relative to $\mathcal{S}$, is depicted in the tree of Figure 4.9(a).

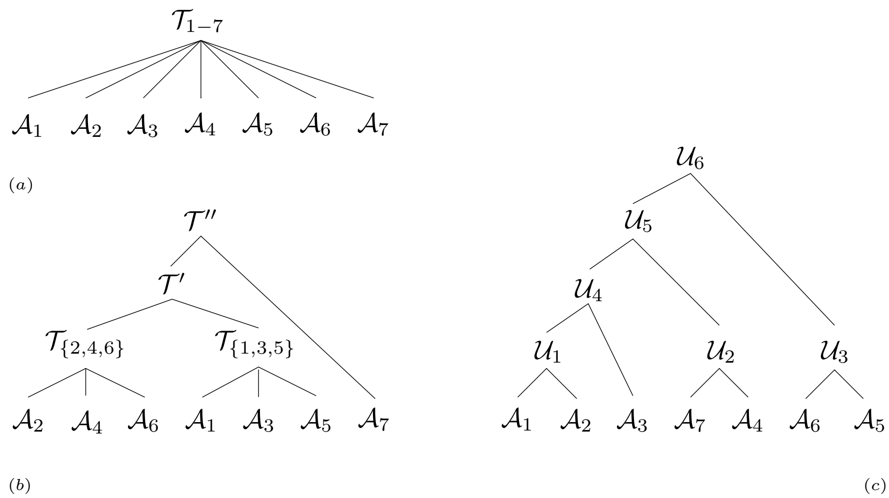


Fig. 4.9. Three synchronized automata constructed from $\{A_i \mid i \in [7]\}$.

Next consider the synchronized automaton $\mathcal{T}_{\{2,4,6\}}$ over $\{\mathcal{A}_2, \mathcal{A}_4, \mathcal{A}_6\}$ and the synchronized automaton $\mathcal{T}_{\{1,3,5\}}$ over $\{\mathcal{A}_1, \mathcal{A}_3, \mathcal{A}_5\}$. Let $\mathcal{T}_{\{2,4,6\}}$ be specified as $\mathcal{T}_{\{2,4,6\}} = (P_1, \Gamma_1, \gamma_1, J_1)$ and let $\mathcal{T}_{\{1,3,5\}}$ be specified as $\mathcal{T}_{\{1,3,5\}} = (P_2, \Gamma_2, \gamma_2, J_2)$.

Let $\mathcal{T}'$ be a synchronized automaton over $\mathcal{S}' = \{\mathcal{A}'_1, \mathcal{A}'_2\}$, with $\mathcal{A}'_1 = \mathcal{T}_{\{2,4,6\}}$ and $\mathcal{A}'_2 = \mathcal{T}_{\{1,3,5\}}$. Let $\mathcal{T}'$ be specified as $\mathcal{T}' = (P', \Gamma', \gamma', J')$.

Let $\mathcal{T}''$ be a synchronized automaton over $\mathcal{S}'' = \{\mathcal{A}''_1, \mathcal{A}''_2\}$, with $\mathcal{A}''_1 = \mathcal{T}'$ and $\mathcal{A}''_2 = \mathcal{A}_7$. Let $\mathcal{T}''$ be specified as $\mathcal{T}'' = (P'', \Gamma'', \gamma'', J'')$, for some $\gamma'' \subseteq P'' \times \Gamma'' \times P''$. By Definition 4.1.2 we have $P'' = P' \times Q_7 = (\prod_{i\in\{1,2\}} P_i) \times$

$Q_7 = ((\prod_{i\in\{2,4,6\}} Q_i) \times (\prod_{i\in\{1,3,5\}} Q_i)) \times Q_7 = ((Q_2 \times Q_4 \times Q_6) \times (Q_1 \times Q_3 \times Q_5)) \times Q_7$. Similarly, $J'' = ((I_2 \times I_4 \times I_6) \times (I_1 \times I_3 \times I_5)) \times I_7$. Furthermore, $\Gamma'' = \Gamma' \cup \Sigma_7 = (\bigcup_{i\in\{1,2\}} \Gamma_i) \cup \Sigma_7 = ((\bigcup_{i\in\{2,4,6\}} \Sigma_i) \cup (\bigcup_{i\in\{1,3,5\}} \Sigma_i)) \cup \Sigma_7 = \bigcup_{i\in[7]} \Sigma_i$.

Thus $\mathcal{T}''$ has the same actions as any synchronized automaton formed directly over $\mathcal{S}$. Its set of states, however, differs from the set of states of a synchronized automaton over $\mathcal{S}$ by its nested structure and its ordering. In Figure 4.9(b) the structure of $\mathcal{T}''$ relative to $\mathcal{S}$ is depicted.

In Figure 4.9(c) the structure relative to $\mathcal{S}$ of yet another route for constructing a synchronized automaton, starting from the automata in $\mathcal{S}$, is depicted. The set of states of this particular synchronized automaton $\mathcal{U}_6$ is $(((Q_1 \times Q_2) \times Q_3) \times (Q_7 \times Q_4)) \times (Q_6 \times Q_5)$. □

In order to describe in a precise way the relationship between a synchronized automaton obtained by iteratively composing synchronized automata and a synchronized automaton formed directly from a given set of automata, we need formal notions enabling us to describe the construction and the parsing of vectors with vectors as elements. Let $\mathcal{D} = \{D_j \mid j \in J\}$ be an indexed set, with $J \subseteq \mathbb{N}$ and $J \neq \varnothing$. Then $\mathcal{V}(\mathcal{D})$ is defined as consisting of all finitely nested combinations of elements from $\mathcal{D}$ provided each D_j is used at most once. The *domain* of an element V from $\mathcal{V}(\mathcal{D})$ consequently is defined to consist of the indices of the sets in $\mathcal{D}$ combined to form V. This leads to the following recursive definition of $\mathcal{V}(\mathcal{D})$ and the accompanying notion of domain.

Definition 4.3.2. $\mathcal{V}(\mathcal{D})$ *is the smallest set* $\mathcal{V}$ *such that*

(1) $D_j \in \mathcal{V}$, *for each* $j \in J$;
Set $dom(D_j) = \{j\}$, *and*

(2) if $\{V_\ell \mid \ell \in L\} \subseteq \mathcal{V}$, *with* $L \subseteq \mathbb{N}$ *and* $L \neq \varnothing$, *then* $\prod_{\ell\in L} V_\ell \in \mathcal{V}$ *provided that for all* $k \neq \ell \in L$, $dom(V_k) \cap dom(V_\ell) = \varnothing$;
Set $dom(\prod_{\ell\in L} V_\ell) = \bigcup_{\ell\in L} dom(V_\ell)$. □

This definition provides a description of how to construct products of products of indexed sets. Every element of $\mathcal{V}(\mathcal{D})$ describes a finitely nested cartesian product of sets from $\mathcal{D}$, while its domain gives the information as to which D_j have been used.

Note that according to step (2) of Definition 4.3.2 each product may combine an infinite number of sets. In the construction of any product in $\mathcal{V}$, however, step (2) is applied only a finite number of times. This corresponds to the intuition that a synchronized automaton is constructed by a finite iteration.

Example 4.3.3. (Example 4.3.1 continued) Let $\mathcal{Q} = \{Q_i \mid i \in [7]\}$. The set of states $P_2 = \prod_{i\in\{1,3,5\}} Q_i$ is an element of $\mathcal{V}(\mathcal{Q})$ with domain $\{1,3,5\}$. Also $P' = P_1 \times P_2 = \prod_{i\in\{2,4,6\}} Q_i \times \prod_{i\in\{1,3,5\}} Q_i$ is an element of $\mathcal{V}(\mathcal{Q})$. Its domain is $\{2,4,6\} \cup \{1,3,5\} = \{1,2,3,4,5,6\}$. Finally, for $P'' = P' \times Q_7 \in \mathcal{V}(\mathcal{Q})$, we have $\mathrm{dom}\,(P' \times Q_7) = \{1,2,3,4,5,6,7\}$. □

Given an element v of a nested cartesian product V from $\mathcal{V}(\mathcal{D})$ with domain $\mathrm{dom}\,(V)$, we want to *unpack* and *reorder* v in such a way that the "corresponding" element of $\prod_{j\in\mathrm{dom}\,(V)} D_j$ results. To this end we define the function u_V which recursively, for each $j \in \mathrm{dom}\,(V)$, locates in v the element in the position of D_j according to the construction of V. Note that since each D_j with $j \in \mathrm{dom}\,(V)$ is used exactly once in the construction of V, its position in V is unique. Thus u_V **u**npacks v and on basis of this unpacking the resulting elements of $\bigcup_{j\in\mathrm{dom}\,(V)} D_j$ are ordered in $\langle v\rangle_V$ according to $\mathrm{dom}\,(V)$.

Definition 4.3.4. *Let $V \in \mathcal{V}(\mathcal{D})$ be such that $\mathrm{dom}\,(V) = J'$ for some $J' \subseteq J$. Then*

(1) the function $u_V : V \times J' \to \bigcup_{j\in J'} D_j$ is defined as follows:
  (a) if $J' = \{j\}$ and $V = D_j$, then $u_V(v,j) = v$ for all $v \in V$ and
  (b) if $V = \prod_{\ell\in L} V_\ell$, with $V_\ell \in \mathcal{V}(\mathcal{D})$ for all $\ell \in L$, then, for all $v \in V$ and $j \in J'$, $u_V(v,j) = u_{V_k}(proj_k(v),j)$, where $k \in L$ is such that $j \in dom\,(V_k)$, and

(2) the reordering *of an element $v \in V$ relative to the construction of V is denoted by $\langle v\rangle_V$ and is defined as*

$$\langle v\rangle_V = \prod_{j\in J'} u_V(v,j). \qquad \square$$

Example 4.3.5. (Example 4.3.3 continued) Assume that we know that $q = (((x,m,\ell),(e,a,p)),e) \in P''$. With the above definition we now reorder q relative to the construction of P'': $\langle q\rangle_{P''} = \prod_{i\in[7]} u_{P''}(q,i)$. Here, e.g., $u_{P''}(q,3) = a$. This follows from the fact that $u_{P''}((((x,m,\ell),(e,a,p)),e),3) = u_{P'}(((x,m,\ell),(e,a,p)),3)$ since $3 \in \mathrm{dom}\,(P')$, $u_{P'}(((x,m,\ell),(e,a,p)),3) = u_{P_2}((e,a,p),3)$ as $3 \in \mathrm{dom}\,(P_2)$, and $u_{P_2}((e,a,p),3) = u_{Q_3}(a,3) = a$. Each $u_{P''}(q,i)$ can thus be determined, leading to $\langle q\rangle_{P''} = (e,x,a,m,p,\ell,e)$. □

Definition 4.3.4 may seem unnecessarily complicated but, as illustrated in the next example, the information about the construction of $V \in \mathcal{V}(\mathcal{D})$ is necessary in order to obtain a faithful reordering of the entries from $\bigcup_{j\in J} D_j$ in $\mathcal{V}$.

Example 4.3.6. Let $\mathcal{Q} = \{Q_i \mid i \in [3]\}$. Let $a \in Q_1$ and let $b, c \in Q_2 \cap Q_3$. Now assume we want to reorder $q = (a, (b, c))$. Then we need to know whether we are dealing with a construction $Q_1 \times (Q_2 \times Q_3) \in \mathcal{V}(\mathcal{Q})$, which would mean that the faithful reordering of q is (a, b, c), or with a construction $Q_1 \times (Q_3 \times Q_2) \in \mathcal{V}(\mathcal{Q})$, which would result in (a, c, b) as the faithful reordering of q. $\square$

Only if $D_i \cap D_j = \varnothing$ for any two sets of states of a set of automata, the above definitions could be simplified. This has never been a condition though.

Unpacking and reordering all elements of a nested cartesian product V over sets from $\mathcal{D}$ (relative to the construction of V) results in the cartesian product (over sets from $\mathcal{D}$) according to J. This is formally stated in the following lemma.

Lemma 4.3.7. *If $V \in \mathcal{V}(\mathcal{D})$ and $dom(V) = J'$, then $\{\langle v \rangle_V \mid v \in V\} = \prod_{j \in J'} D_j$.*

Proof. Let $V \in \mathcal{V}(\mathcal{D})$ and let $\mathrm{dom}\,(V) = J'$.

($\subseteq$) Let $v \in V$. By Definition 4.3.4 we have $\langle v \rangle_V = \prod_{j \in J'} u_V(v, j)$. Now we only have to prove that $u_V(v, j) \in D_j$, for all $j \in J'$. We do this by structural induction.
If $J' = \{j\}$ and $V = D_j$, then $u_V(v, j) = v \in V = D_j$.
Next assume that $V = \prod_{\ell \in L} V_\ell$, with $V_\ell \in \mathcal{V}(\mathcal{D})$ for all $\ell \in L$. Then, by Definition 4.3.4, for all $j \in J'$, $u_V(v, j) = u_{V_k}(\mathrm{proj}_k(v), j)$, where k is such that $j \in \mathrm{dom}\,(V_k)$. Since each $V_k \in \mathcal{V}(\mathcal{D})$, the depth of its nesting is strictly less than the depth of the nesting in V. Thus by the induction hypothesis, $u_{V_k}(\mathrm{proj}_k(v), j) \in D_j$, for all $j \in \mathrm{dom}\,(V_k)$, which completes this direction of the proof.

($\supseteq$) Let $d \in \prod_{j \in J'} D_j$. Then we only have to prove that there exists a $v \in V$ such that $\langle v \rangle_V = d$ or, equivalently, that there exists a $v \in V$ such that for all $j \in J'$, $u_V(v, j) = \mathrm{proj}_j(d)$. We do this by structural induction.
Assume that $J' = \{j\}$ and $V = D_j$. Now set $v = \mathrm{proj}_j(d)$. Then $u_V(v, j) = v = \mathrm{proj}_j(d)$.
Next assume that $V = \prod_{\ell \in L} V_\ell$. Then from the induction hypothesis it follows that for all $\ell \in L$, $\{\langle v_\ell \rangle_{V_\ell} \mid v_\ell \in V_\ell\} = \prod_{j \in J_\ell} D_j$ where $J_\ell = \mathrm{dom}\,(V_\ell)$. Hence for all $\ell \in L$ and for all $j \in J_\ell$ we have a $v_\ell \in V_\ell$ such that $u_{V_\ell}(v_\ell, j) = \mathrm{proj}_j(d) \in D_j$. Let $v \in V$ be such that for all $\ell \in L$, $\mathrm{proj}_\ell(v) = v_\ell$ with $v_\ell \in V_\ell$. Then for all $j \in J'$, $u_V(v, j) = u_{V_\ell}(\mathrm{proj}_\ell(v), j)$, where ℓ is such that $j \in \mathrm{dom}\,(V_\ell)$. Since for all $\ell \in L$, $u_{V_\ell}(\mathrm{proj}_\ell(v), j) = u_{V_\ell}(v_\ell, j) = \mathrm{proj}_j(d)$, this completes also this direction of the proof. $\square$

Now we are ready to return to the issue of iteratively forming a synchronized automaton, given a set of synchronized automata. We begin by generalizing the notion of a synchronized automaton.

Definition 4.3.8. *$\mathcal{T}$ is an* iterated synchronized automaton over *$\mathcal{S}$ if either*

(1) $\mathcal{T}$ is a synchronized automaton over $\mathcal{S}$, or

(2) $\mathcal{T}$ is a synchronized automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where each $\mathcal{T}_j$ is an iterated synchronized automaton over $\{\mathcal{A}_i \mid i \in \mathcal{I}_j\}$, for some $\mathcal{I}_j \subseteq \mathcal{I}$, and $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$. □

We see that iterated synchronized automata indeed are a generalization of synchronized automata: every synchronized automaton over a given set of automata may also be viewed as an iterated synchronized automaton over that set. But, as announced in the beginning of this section, synchronized automata formed iteratively over a set of automata are essentially synchronized automata over that set. Intuitively the only difference lies in the ordering and grouping of the elements from the set of automata. In the remainder of this section, we will formalize this statement.

The following lemma shows that the set of (initial) states of an iterated synchronized automaton over a set of automata is — upto a reordering — the same as the set of (initial) states of any synchronized automaton over that set.

Lemma 4.3.9. *Let $\mathcal{T} = (P, \Gamma, \gamma, J)$ be an iterated synchronized automaton over $\mathcal{S}$. Let $\mathcal{Q} = \{Q_i \mid i \in \mathcal{I}\}$. Then*

(1) $P \in \mathcal{V}(\mathcal{Q})$ and $dom(P) = \mathcal{I}$,

(2) $\{\langle q \rangle_P \mid q \in P\} = \prod_{i\in\mathcal{I}} Q_i$, and

(3) $\{\langle q \rangle_P \mid q \in J\} = \prod_{i\in\mathcal{I}} I_i$.

Proof. If $\mathcal{T}$ is a synchronized automaton over $\mathcal{S}$, then $P = \prod_{i\in\mathcal{I}} Q_i$ and $J = \prod_{i\in\mathcal{I}} I_i$.
By Definition 4.3.2(2) we have $P \in \mathcal{V}(\mathcal{Q})$ and $\mathrm{dom}\,(P) = \bigcup_{i\in\mathcal{I}} \mathrm{dom}\,(Q_i) = \mathcal{I}$.
By Lemma 4.3.7 we have $\{\langle q \rangle_P \mid q \in P\} = \prod_{i\in\mathcal{I}} Q_i$.
Since according to Definition 4.3.4 for all $q \in P$, $\langle q \rangle_P = \prod_{i\in\mathcal{I}} u_P(q,i) = \prod_{i\in\mathcal{I}} u_{Q_i}(\mathrm{proj}_i(q), i) = \prod_{i\in\mathcal{I}} \mathrm{proj}_i(q) = q$, it follows that $\{\langle q \rangle_P \mid q \in J\} = \{q \mid q \in \prod_{i\in\mathcal{I}} I_i\} = \prod_{i\in\mathcal{I}} I_i$.

Now assume that $\mathcal{T}$ is an iterated synchronized automaton over $\mathcal{S}$. Hence $\mathcal{T}$ is a synchronized automaton over a set of automata $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$, and each $\mathcal{T}_j$ is an iterated synchronized automaton over $\{\mathcal{A}_i \mid i \in \mathcal{I}_j\}$. Let, for $j \in \mathcal{J}$, $\mathcal{T}_j$ be specified as $\mathcal{T}_j = (P_j, \Gamma_j, \gamma_j, J_j)$. Hence $P = \prod_{j\in\mathcal{J}} P_j$ and $J = \prod_{j\in\mathcal{J}} J_j$. As induction hypothesis we assume that for all $j \in \mathcal{J}$, $P_j \in \mathcal{V}(\mathcal{Q})$ with $\mathrm{dom}\,(P_j) = \mathcal{I}_j$, and

$\{\langle q\rangle_{P_j} \mid q \in J_j\} = \prod_{i\in\mathcal{I}_j} I_i$.
Since $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$, we immediately have $P = \prod_{j\in\mathcal{J}} P_j \in \mathcal{V}(\mathcal{Q})$ and $\mathrm{dom}\,(P) = \bigcup_{j\in\mathcal{J}} \mathrm{dom}\,(P_j) = \bigcup_{j\in\mathcal{J}} \mathcal{I}_j = \mathcal{I}$.
By Lemma 4.3.7 we have $\{\langle q\rangle_P \mid q \in P\} = \prod_{i\in\mathcal{I}} Q_i$.
Furthermore, $q \in J$ if and only if $\mathrm{proj}_j(q) \in J$, for all $j \in \mathcal{J}$. By the induction hypothesis, for all $j \in \mathcal{J}$, $\mathrm{proj}_j(q) \in J_j$ if and only if $\langle \mathrm{proj}_j(q)\rangle_{P_j} = \prod_{i\in\mathcal{I}_j} u_{P_j}(\mathrm{proj}_j(q), i) \in \prod_{i\in\mathcal{I}_j} I_i$. Thus $q \in J$ if and only if for all $j \in \mathcal{J}$ and for all $i \in \mathcal{I}_j$, $u_{P_j}(\mathrm{proj}_j(q), i) \in I_i$. Since for all $q \in P$, $\langle q\rangle_P = \prod_{i\in\mathcal{I}} u_P(q,i) = \prod_{i\in\mathcal{I}} u_{P_{k_i}}(\mathrm{proj}_{k_i}(q), i)$, where $k_i \in \mathcal{J}$ is such that $i \in \mathrm{dom}\,(P_{k_i})$, it follows that $\{\langle q\rangle_P \mid q \in J\} = \prod_{i\in\mathcal{I}} I_i$. □

Next we consider the actions and transitions of iterated synchronized automata. The actions of an iterated synchronized automaton over a set of automata $\mathcal{S}$ are the same as the actions of any synchronized automaton over $\mathcal{S}$. Furthermore, the transitions of any synchronized automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ are — after reordering — the transitions of a synchronized automaton over $\mathcal{S}$.

Lemma 4.3.10. *Let $\mathcal{T} = (P, \Gamma, \gamma, J)$ be an iterated synchronized automaton over $\mathcal{S}$. Then*

(1) $\Gamma = \bigcup_{i\in\mathcal{I}} \Sigma_i$ and

(2) $\{(\langle q\rangle_P, \langle q'\rangle_P) \mid (q,q') \in \gamma_a\} \subseteq \Delta_a(\mathcal{S})$, for all $a \in \Gamma$.

Proof. If $\mathcal{T}$ is a synchronized automaton over $\mathcal{S}$, then (1) follows immediately from Definition 4.1.2. In that case also (2) follows immediately from Definition 4.1.2 because, as in the proof of Lemma 4.3.9, $\langle q\rangle_P = q$, for all $q \in P$.

Now assume that $\mathcal{T}$ is a synchronized automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, and each $\mathcal{T}_j = (P_j, \Gamma_j, \gamma_j, J_j)$ is an iterated synchronized automaton over $\{\mathcal{A}_i \mid i \in \mathcal{I}_j\}$, with $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forming a partition of $\mathcal{I}$. Assume furthermore inductively that for all $j \in \mathcal{J}$, $\Gamma_j = \bigcup_{i\in\mathcal{I}_j} \Sigma_i$. Then $\Gamma = \bigcup_{j\in\mathcal{J}} \Gamma_j = \bigcup_{j\in\mathcal{J}} \bigcup_{i\in\mathcal{I}_j} \Sigma_i = \bigcup_{i\in\mathcal{I}} \Sigma_i$, by Definition 4.1.2, and because $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$.

Consequently we consider the transitions of $\mathcal{T}$. Let $a \in \Gamma$. Since $\mathcal{T}$ is a synchronized automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, we know that $\gamma_a \subseteq \Delta_a(\{\mathcal{T}_j \mid j \in \mathcal{J}\})$. We have to prove that — upto the reordering relative to the construction of P — every a-transition of $\mathcal{T}$ is an element of the complete transition space of a in $\mathcal{S}$. In order to prove this we make inductively the following assumption. For all $j \in \mathcal{J}$, $\{(\langle p\rangle_{P_j}, \langle p'\rangle_{P_j}) \mid (p,p') \in \gamma_{j,a}\} \subseteq \Delta_a(\{\mathcal{A}_i \mid i \in \mathcal{I}_j\})$.

Before we turn to the proof we make the following auxiliary observation. Let $q \in P$. By Lemma 4.3.9 we have $\langle q\rangle_P \in \prod_{i\in\mathcal{I}} Q_i$ and thus $\langle q\rangle_P =$

$\prod_{i\in\mathcal{I}} \mathrm{proj}_i(\langle q\rangle_P)$. Let $i \in \mathcal{I}$. By Definition 4.3.4 we have $\mathrm{proj}_i(\langle q\rangle_P) = u_P(q,i) = u_{P_j}(\mathrm{proj}_j(q),i)$, where j is such that $i \in \mathcal{I}_j$. Now $\mathrm{proj}_j(q) \in P_j$ and hence, again by Lemma 4.3.9, $\langle \mathrm{proj}_j(q)\rangle_{P_j} \in \prod_{i\in\mathcal{I}_j} Q_i$. By Definition 4.3.4 once again we have $\mathrm{proj}_i(\langle \mathrm{proj}_j(q)\rangle_{P_j}) = u_{P_j}(\mathrm{proj}_j(q),i)$, whenever $i \in \mathcal{I}_j$. Hence $\mathrm{proj}_i(\langle q\rangle_P) = \mathrm{proj}_i(\langle \mathrm{proj}_j(q)\rangle_{P_j})$, for all $q \in P$, $i \in \mathcal{I}_j$, and $j \in \mathcal{J}$. This ends the observation.

Now let $(q,q') \in \gamma_a$. In order to prove that $(\langle q\rangle_P, \langle q'\rangle_P) \in \Delta_a(\mathcal{S})$ we verify the two conditions in Definition 4.1.1.
First we prove that there exists an $i \in \mathcal{I}$ such that $\mathrm{proj}_i^{[2]}(\langle q\rangle_P, \langle q'\rangle_P) \in \delta_{i,a}$. Let $j \in \mathcal{J}$ be such that $\mathrm{proj}_j^{[2]}(q,q') \in \gamma_{j,a}$. Such a j exists because $\gamma_a \subseteq \Delta_a(\{\mathcal{T}_j \mid j \in \mathcal{J}\})$. By the induction hypothesis we have $(\langle \mathrm{proj}_j(q)\rangle_{P_j}, \langle \mathrm{proj}_j(q')\rangle_{P_j}) \in \Delta_a(\{\mathcal{A}_i \mid i \in \mathcal{I}_j\})$. Hence by Definition 4.1.1 there exists an $i \in \mathcal{I}_j$ such that $\mathrm{proj}_i^{[2]}(\langle \mathrm{proj}_j(q)\rangle_{P_j}, \langle \mathrm{proj}_j(q')\rangle_{P_j}) \in \delta_{i,a}$. Thus, by our observation above, for this i we have $\mathrm{proj}_i^{[2]}(\langle q\rangle_P, \langle q'\rangle_P) \in \delta_{i,a}$, as desired.
Secondly, we prove that for all $i \in \mathcal{I}$, either $\mathrm{proj}_i^{[2]}(\langle q\rangle_P, \langle q'\rangle_P) \in \delta_{i,a}$ or $\mathrm{proj}_i(\langle q\rangle_P) = \mathrm{proj}_i(\langle q'\rangle_P)$. Let $i \in \mathcal{I}$ and let $j \in \mathcal{J}$ be such that $i \in \mathcal{I}_j$. Because $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$ such a j exists and is unique. Since $\gamma_a \subseteq \Delta_a(\{\mathcal{T}_j \mid j \in \mathcal{J}\})$, Definition 4.1.1 implies that either $\mathrm{proj}_j^{[2]}(q,q') \in \gamma_{j,a}$ or $\mathrm{proj}_j(q) = \mathrm{proj}_j(q')$.
If $\mathrm{proj}_j^{[2]}(q,q') \in \gamma_{j,a}$, then $(\langle \mathrm{proj}_j(q)\rangle_{P_j}, \langle \mathrm{proj}_j(q')\rangle_{P_j}) \in \Delta_a(\{\mathcal{A}_i \mid i \in \mathcal{I}_j\})$ by the induction hypothesis. Hence by Definition 4.1.1, we get that either $\mathrm{proj}_i^{[2]}(\langle \mathrm{proj}_j(q)\rangle_{P_j}, \langle \mathrm{proj}_j(q')\rangle_{P_j}) \in \delta_{i,a}$, which — by the above auxiliary observation — implies that $\mathrm{proj}_i^{[2]}(\langle q\rangle_P, \langle q'\rangle_P) \in \delta_{i,a}$, or $\mathrm{proj}_i(\langle \mathrm{proj}_j(q)\rangle_{P_j}) = \mathrm{proj}_i(\langle \mathrm{proj}_j(q')\rangle_{P_j})$, which — again by the above auxiliary observation — implies that $\mathrm{proj}_i(\langle q\rangle_P) = \mathrm{proj}_i(\langle q'\rangle_P)$.
If $\mathrm{proj}_j(q) = \mathrm{proj}_j(q')$, then $\mathrm{proj}_i(\langle q\rangle_P) = u_{P_j}(\mathrm{proj}_j(q),i) = u_{P_j}(\mathrm{proj}_j(q'),i)$ $= \mathrm{proj}_i(\langle q'\rangle_P)$, which completes the proof. □

Note that this lemma states that for each action a its complete transition space in $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is included — after reordering — in its complete transition space in $\mathcal{S}$. Iteration in the construction of a synchronized automaton thus does not lead to an increase of the number of possibilities for synchronization. In other words, every iterated synchronized automaton over a set of automata can be interpreted as a synchronized automaton over that set, by reordering its state space and transition space.

Definition 4.3.11. *Let $\mathcal{T} = (Q,\Sigma,\delta,I)$ be an iterated synchronized automaton over $\mathcal{S}$. Then the* reordered version of $\mathcal{T}$ w.r.t. $\mathcal{S}$ *is denoted by $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$ and is defined as*

$$\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}} = (\{\langle q\rangle_Q \mid q \in Q\}, \Sigma, \\ \{(\langle q\rangle_Q, a, \langle q'\rangle_Q) \mid q, q' \in Q,\ (q, a, q') \in \delta\}, \{\langle q\rangle_I \mid q \in I\}). \ \square$$

From Lemmata 4.3.9 and 4.3.10 we conclude that $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$ is indeed a synchronized automaton over $\mathcal{S}$ whenever $\mathcal{T}$ is an iterated synchronized automaton over $\mathcal{S}$. In fact, $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$ is the interpretation of $\mathcal{T}$ as a synchronized automaton over $\mathcal{S}$ by reordering. Since their only difference is the ordering of the elements of their state spaces, it is immediate that $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$ and $\mathcal{T}$ have — upto a reordering — the same set of computations and thus the same behavior.

Theorem 4.3.12. *Let $\mathcal{T} = (Q, \Sigma, \delta, I)$ be an iterated synchronized automaton over $\mathcal{S}$ and let Θ be an alphabet disjoint from Q. Then*

(1) $\mathbf{C}^{\infty}_{\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}} = \{\langle q_0\rangle_Q a_1 \langle q_1\rangle_Q a_2 \langle q_2\rangle_Q \cdots \mid q_0 a_1 q_1 a_2 q_2 \cdots \in \mathbf{C}^{\infty}_{\mathcal{T}}\}$ *and*

(2) $\mathbf{B}^{\Theta,\infty}_{\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}} = \mathbf{B}^{\Theta,\infty}_{\mathcal{T}}$. $\square$

Clearly the converse of the inclusion of Lemma 4.3.10(2) in general does not hold, since synchronized automata — and hence also iterated synchronized automata — are equipped with only a subset of all possible synchronizations. Moreover, a given intermediate synchronized automaton $\mathcal{T}_j$ over a subset $\mathcal{S}_j$ of $\mathcal{S}$ may have a transition relation that is properly included in the complete transition space of $\mathcal{S}_j$. As a consequence, $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ may provide less transitions for the forming of a synchronized automaton than $\{\mathcal{A}_i \mid i \in \mathcal{I}\}$ does. However, there is a natural condition that guarantees that for a given arbitrary synchronized automaton $\mathcal{T}$ over $\mathcal{S}$ and given iterated synchronized automata $\mathcal{T}_j$ over subsets $\mathcal{S}_j = \{\mathcal{A}_i \mid i \in \mathcal{I}_j\}$, where the $\mathcal{I}_j$ form a partition of $\mathcal{I}$, one can still obtain a synchronized automaton $\widehat{\mathcal{T}}$ over the set consisting of the $\mathcal{T}_j$, such that $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_{\mathcal{S}} = \mathcal{T}$. This condition requires that each of the $\mathcal{T}_j$ has at least all transitions — after reordering — of the corresponding subautomaton of $\mathcal{T}$ determined by $\mathcal{I}_j$. In fact, when loops are ignored this is a necessary and sufficient condition for obtaining an iterated version of a given synchronized automaton over $\mathcal{S}$. Formally, we have the following result, where we recall $\delta_{\mathcal{I}_j}$ to be the transition relation of $SUB_{\mathcal{I}_j}(\mathcal{T})$.

Theorem 4.3.13. *Let $\mathcal{T} = (Q, \Sigma, \delta, I)$ be a synchronized automaton over $\mathcal{S}$ and let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, form a partition of $\mathcal{I}$. Let, for each $j \in \mathcal{J}$, $\mathcal{T}_j = (P_j, \Gamma_j, \gamma_j, J_j)$ be an iterated synchronized automaton over $\{\mathcal{A}_i \mid i \in \mathcal{I}_j\}$. Then*

(1) if $(\delta_{\mathcal{I}_j})_a \subseteq \{(\langle q\rangle_{P_j}, \langle q'\rangle_{P_j}) \mid (q, q') \in \gamma_{j,a}\}$, for all $a \in \Gamma_j$ for all $j \in \mathcal{J}$, then there exists a synchronized automaton $\widehat{\mathcal{T}}$ over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ such that $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_{\mathcal{S}} = \mathcal{T}$, and

(2) if $\widehat{\mathcal{T}}$ is a synchronized automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, then $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_{\mathcal{S}} = \mathcal{T}$ implies that $(\delta_{\mathcal{I}_j})_a \setminus \{(p,p) \mid (p,p) \in \Delta_a(\{\mathcal{A}_i \mid i \in \mathcal{I}_j\})\} \subseteq \{(\langle q\rangle_{P_j}, \langle q'\rangle_{P_j}) \mid (q,q') \in \gamma_{j,a}\}$, for all $a \in \Gamma_j$ for all $j \in \mathcal{J}$.

Proof. Let $\widehat{\mathcal{T}} = (P, \Gamma, \gamma, J)$ be an arbitrary synchronized automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$. First we make an auxiliary observation similar to the one in the proof of Lemma 4.3.10. Let $q \in P$ and let $j \in \mathcal{J}$. Then $\mathrm{proj}_{\mathcal{I}_j}(\langle q\rangle_P) = \langle \mathrm{proj}_j(q)\rangle_{P_j}$, since $P = \prod_{j\in\mathcal{J}} P_j$ and, by Lemma 4.3.9(2), $\prod_{i\in\mathcal{I}_j} Q_i = \{\langle q\rangle_{P_j} \mid q \in P_j\}$.

(1) Assume that $(\delta_{\mathcal{I}_j})_a \subseteq \{(\langle q\rangle_{P_j}, \langle q'\rangle_{P_j}) \mid (q,q') \in \gamma_{j,a}\}$. By Lemmata 4.3.9(2), 4.3.10(1), and 4.3.9(3) we know that $Q = \{\langle q\rangle_P \mid q \in P\}$, $\Sigma = \Gamma$, and $I = \{\langle q\rangle_J \mid q \in J\}$, respectively. Thus it only remains to prove that the transition relation γ for $\widehat{\mathcal{T}}$ can be chosen in such a way that $\delta = \{(\langle q\rangle_P, a, \langle q'\rangle_P) \mid q, q' \in P,\ (q,a,q') \in \gamma\}$. Thus using the injectivity of reordering we define γ simply by $\gamma_a = \{(q,q') \in \prod_{j\in\mathcal{J}} P_j \times \prod_{j\in\mathcal{J}} P_j \mid (\langle q\rangle_P, \langle q'\rangle_P) \in \delta_a\}$, for all $a \in \Gamma$ and prove that this is indeed the transition relation of a synchronized automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$.

Let $(p,p') \in \gamma_a$. We prove there exists a $j \in \mathcal{J}$ so that $\mathrm{proj}_j^{[2]}(p,p') \in \gamma_{j,a}$. As $(\langle p\rangle_P, \langle p'\rangle_P) \in \delta_a$ there exists an $i \in \mathcal{I}$ such that $\mathrm{proj}_j^{[2]}(\langle p\rangle_P, \langle p'\rangle_P) \in \delta_{i,a}$. Let j be such that $i \in \mathcal{I}_j$. Then it follows that $\mathrm{proj}_{\mathcal{I}_j}^{[2]}(\langle p\rangle_P, \langle p'\rangle_P) \in (\delta_{\mathcal{I}_j})_a$. Since $(\delta_{\mathcal{I}_j})_a \subseteq \{(\langle q\rangle_{P_j}, \langle q'\rangle_{P_j}) \mid (q,q') \in \gamma_{j,a}\}$ there exists an $(r,r') \in \gamma_{j,a}$ such that $(\langle r\rangle_{P_j}, \langle r'\rangle_{P_j}) = \mathrm{proj}_{\mathcal{I}_j}^{[2]}(\langle p\rangle_P, \langle p'\rangle_P)$. Thus by the observation above we have $(\langle r\rangle_{P_j}, \langle r'\rangle_{P_j}) = (\langle \mathrm{proj}_j(p)\rangle_{P_j}, \langle \mathrm{proj}_j(p')\rangle_{P_j})$. Since reordering is an injective operation it follows that $r = \mathrm{proj}_j(p)$ and $r' = \mathrm{proj}_j(p')$, and thus $\mathrm{proj}_j^{[2]}(p,p') = (r,r') \in \gamma_{j,a}$.

It now remains to prove that for all $j \in \mathcal{J}$, either $\mathrm{proj}_j(p) = \mathrm{proj}_j(p')$ or $\mathrm{proj}_j^{[2]}(p,p') \in \gamma_{j,a}$. Let $j \in \mathcal{J}$ be such that $\mathrm{proj}_j(p) \neq \mathrm{proj}_j(p')$. Then we only have to prove that $\mathrm{proj}_j^{[2]}(p,p') \in \gamma_{j,a}$. Since $(p,p') \in \gamma_a$ we have $(\langle p\rangle_P, \langle p'\rangle_P) \in \delta_a$. By the observation above we have $\mathrm{proj}_{\mathcal{I}_j}(\langle p\rangle_P) = \langle \mathrm{proj}_j(p)\rangle_{P_j}$ and $\mathrm{proj}_{\mathcal{I}_j}(\langle p'\rangle_P) = \langle \mathrm{proj}_j(p')\rangle_{P_j}$. From the fact that reordering is an injective operation we infer that $\mathrm{proj}_{\mathcal{I}_j}(\langle p\rangle_P) \neq \mathrm{proj}_{\mathcal{I}_j}(\langle p'\rangle_P)$. Hence $\mathrm{proj}_{\mathcal{I}_j}^{[2]}(\langle p\rangle_P, \langle p'\rangle_P) \in (\delta_{\mathcal{I}_j})_a$. Since $(\delta_{\mathcal{I}_j})_a \subseteq \{(\langle q\rangle_{P_j}, \langle q'\rangle_{P_j}) \mid (q,q') \in \gamma_{j,a}\}$ it follows that $\mathrm{proj}_j^{[2]}(p,p') \in \gamma_{j,a}$.

(2) Now assume that $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_{\mathcal{S}} = \mathcal{T}$. Let $j \in \mathcal{J}$ and $a \in \Gamma$ be fixed. Let $(p,p') \in (\delta_{\mathcal{I}_j})_a$ be such that $p \neq p'$. By Definition 4.1.6 there is a pair $(r,r') \in \delta_a$ such that $\mathrm{proj}_{\mathcal{I}_j}^{[2]}(r,r') = (p,p')$. Since $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_{\mathcal{S}} = \mathcal{T}$ there are $(\hat{r},\hat{r}') \in \gamma_a$ such that $(\langle\hat{r}\rangle_P, \langle\hat{r}'\rangle_P) = (r,r')$. By the observation above we have $(p,p') = \mathrm{proj}_{\mathcal{I}_j}^{[2]}(r,r') = (\langle \mathrm{proj}_j(\hat{r})\rangle_{P_j}, \langle \mathrm{proj}_j(\hat{r}')\rangle_{P_j})$ and thus the only thing left to prove here is that $(\mathrm{proj}_j(\hat{r}), \mathrm{proj}_j(\hat{r}')) \in \gamma_{j,a}$. Assume to the contrary that this is not the case. Then the fact that $\widehat{\mathcal{T}}$ is a synchronized automaton

over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, together with $(\hat{r}, \hat{r}') \in \gamma_a$, implies that $\mathrm{proj}_j(\hat{r}) = \mathrm{proj}_j(\hat{r}')$ and thus $p = p'$, a contradiction. Hence $(\mathrm{proj}_j(\hat{r}), \mathrm{proj}_j(\hat{r}')) \in \gamma_{j,a}$. □

Thus, not only can every iterated synchronized automaton over $\mathcal{S}$ be considered as a synchronized automaton directly constructed from $\mathcal{S}$ by Definition 4.3.11, but according to Theorem 4.3.13 also every synchronized automaton can be iteratively constructed from its subautomata. Consequently, both subautomata and iterated synchronized automata can be treated as synchronized automata — including the considerations concerning their computations and behavior — and it thus suffices to study only the relationship between (sub)automata and synchronized automata in the sequel, i.e. without considering iterated synchronized automata explicitly.

4.4 Synchronizations

As said before, the high level of flexibility that is obtained by leaving the set of transitions of a synchronized automaton as a modeling choice is an important — perhaps even the most important — feature of the team automata framework we are introducing. The choice for a specific interconnection strategy (which automata synchronize on what actions, and when) is based on the system one wants to model.

In this section we provide the basis for the introduction of a broad variety of often complex interconnection strategies for team automata in Section 5.3. We do so by introducing some basic and natural types of synchronization that can be expressed already within the synchronized automata underlying team automata.

We focus on the individual actions of a synchronized automaton and we distinguish several different ways of synchronizing on shared actions. We consider actions that are never used in synchronizations between multiple automata, as well as actions on which all automata having these actions have to synchronize. The latter case is weakened by requiring participation only if an automaton is in a state at which that action is enabled.

Recall that information on the actual execution of loops is missing in the transition relation of a synchronized automaton. In the coming definitions and their intuitive explanation, the presence of loops on action a in automata is treated as if a is actually executed, which is in accordance with the maximal interpretation of the participation of automata adopted in Section 4.2.

Notation 2. *For the remainder of this chapter we assume $\mathcal{T} = (Q, \Sigma, \delta, I)$ is an arbitrary but fixed synchronized automaton over our fixed set $\mathcal{S}$ of au-*

tomata. Note that Σ is the alphabet of any synchronized automaton over $\mathcal{S}$ (i.e. not only of $\mathcal{T}$). □

4.4.1 Free

Intuitively, an action a is a *free* action of $\mathcal{T}$ if no a-transition of $\mathcal{T}$ is brought about by a simultaneous execution of a by two or more automata. Thus, whenever a is executed by $\mathcal{T}$ only one automaton is active in this execution.

Definition 4.4.1. *The set of* free *actions of $\mathcal{T}$ is denoted by $Free(\mathcal{T})$ and is defined as*

$$Free(\mathcal{T}) = \{a \in \Sigma \mid (q,q') \in \delta_a \Rightarrow \\ \#\{i \in \mathcal{I} \mid a \in \Sigma_i \wedge proj_i^{[2]}(q,q') \in \delta_{i,a}\} = 1\}. \;\square$$

Example 4.4.2. (Example 4.1.3 continued) Actions a and b both are not *free* in synchronized automaton $\mathcal{T}_{\{1,2\}}$. This can be concluded from the fact that the a-transition $((s_1,s_2),a,(t_1,t_2))$ and the b-transition $((t_1,t_2),b,(s_1,s_2))$ can serve as an example of a simultaneous execution of a and b, respectively, by two automata. In synchronized automaton $\mathcal{T}'_{\{1,2\}}$, however, action a is *free* while action b is not *free*. □

4.4.2 Action-Indispensable

If an action a is *action-indispensable*, then all automata which have a as one of their actions are involved in every execution of a by $\mathcal{T}$. This means that $\mathcal{T}$ cannot execute an a if there is an automaton to which a belongs but in which it is not enabled at the current local state.

Definition 4.4.3. *The set of* action-indispensable *(ai for short) actions of $\mathcal{T}$ is denoted by $AI(\mathcal{T})$ and is defined as*

$$AI(\mathcal{T}) = \{a \in \Sigma \mid \forall i \in \mathcal{I} : (a \in \Sigma_i \wedge (q,q') \in \delta_a) \Rightarrow \\ proj_i^{[2]}(q,q') \in \delta_{i,a}\}. \;\square$$

Example 4.4.4. (Example 4.4.2 continued) Actions a and b both are *ai* in the synchronized automaton $\mathcal{T}_{\{1,2\}}$. This follows directly from the fact that in all of the a-transitions and in all of the b-transitions of $\mathcal{T}_{\{1,2\}}$, both W_1 and W_2 participate. Hence b is also *ai* in $\mathcal{T}'_{\{1,2\}}$, while a however is not *ai* in $\mathcal{T}'_{\{1,2\}}$. This difference stems from the fact that in the a-transition $((s_1,s_2),a,(s_1,t_2))$ only W_2 participates while also W_1 has a in its alphabet. □

4.4.3 State-Indispensable

State-indispensable, finally, is a weak version of action-indispensable: if an action a is state-indispensable, then all executions of a by $\mathcal{T}$ involve all automata in which a is enabled at the current local state. In this case $\mathcal{T}$ does not have to "wait" with the execution of a until a is enabled in all automata to which it belongs.

Definition 4.4.5. *The set of* state-indispensable *(si for short) actions of* $\mathcal{T}$ *is denoted by* $SI(\mathcal{T})$ *and is defined as*

$$SI(\mathcal{T}) = \{a \in \Sigma \mid \forall i \in \mathcal{I} : (a \in \Sigma_i \wedge (q, q') \in \delta_a \wedge a \ en_{\mathcal{A}_i} \ proj_i(q)) \Rightarrow proj_i^{[2]}(q, q') \in \delta_{i,a}\}. \ \square$$

Example 4.4.6. (Example 4.4.4 continued) Actions a and b both are *si* in the synchronized automaton $\mathcal{T}_{\{1,2\}}$. This follows immediately from the fact that in all of the a-transitions as well as in all of the b-transitions of $\mathcal{T}_{\{1,2\}}$, both W_1 and W_2 participate. Hence b is also *si* in $\mathcal{T}'_{\{1,2\}}$, whereas a is not *si* in $\mathcal{T}'_{\{1,2\}}$. This is due to the fact that in the a-transition $((s_1, s_2), a, (s_1, t_2))$ only W_2 participates, while at state (s_1, s_2) action a is also enabled at the local state s_1 of W_1. $\square$

4.4.4 Free, Action-Indispensable, and State-Indispensable

We now compare the three types of synchronization introduced in this section.

It is immediate that all *ai* actions in $\mathcal{T}$ also satisfy the weaker requirement of being *si* actions.

Lemma 4.4.7. $AI(\mathcal{T}) \subseteq SI(\mathcal{T})$.

In fact, as we show next, this lemma describes the only dependency among *free*, *ai*, and *si* actions.

The combination of the properties of being *free*, *ai*, and *si* leads in principle to eight different types of actions in a synchronized automaton. However, by Lemma 4.4.7, *ai* implies *si*, which eliminates the combinations ⟨*free*, *ai*, not *si*⟩ and ⟨not *free*, *ai*, not *si*⟩. Each of the remaining six combinations is feasible, as we demonstrate in the following example.

Example 4.4.8. Consider the automata $\mathcal{A}_1 = (\{q, q'\}, \{a\}, \{(q, a, q')\}, \{q\})$ and $\mathcal{A}_2 = (\{r, r'\}, \{a\}, \{(r, a, r')\}, \{r\})$, as depicted in Figure 4.10.

From $\{\mathcal{A}_1, \mathcal{A}_2\}$ we construct the following five synchronized automata $\mathcal{T}^i = (\{(q, r), (q, r'), (q', r), (q', r')\}, \{a\}, \delta^i, \{(q, r)\})$, with $i \in [5]$, where

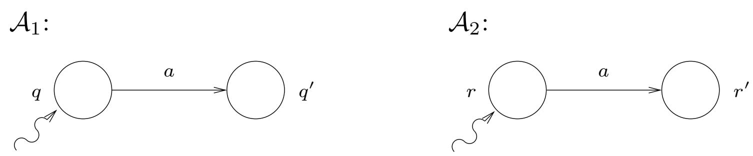


Fig. 4.10. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

δ^1 $= \{((q,r),a,(q,r')),((q,r),a,(q',r'))\}$; now a is not *free* since both automata execute a in the second transition, while a is not *si* (and thus also not *ai*) since $\mathcal{A}_1$ does not execute a in the first transition, even though it is in a state at which a is enabled,

δ^2 $= \{((q,r),a,(q',r'))\}$; now a is not *free* since in the given transition a is executed by both automata, which implies that a is *ai* and thus *si*,

δ^3 $= \{((q,r),a,(q',r))\}$; now a is *free* since only one automaton is involved in the a-transition, but a is not *si* (and thus also not *ai*) since $\mathcal{A}_2$ does not execute a even though it is in a state at which a is enabled,

δ^4 $= \{((q,r'),a,(q',r'))\}$; now a is *free* for the same reason as in the previous case, a is not *ai* since $\mathcal{A}_2$ does have a in its alphabet but nevertheless does not execute a, and a is *si* since $\mathcal{C}_2$ cannot execute a in state r' (a is not enabled at state r'), and

δ^5 $= \varnothing$; now a trivially is *free*, *ai*, and *si*.

These synchronized automata $\mathcal{T}^1$, $\mathcal{T}^2$, $\mathcal{T}^3$, $\mathcal{T}^4$, and $\mathcal{T}^5$ thus illustrate the cases $\langle$not *free*, not *ai*, not *si*$\rangle$, $\langle$not *free*, *ai*, *si*$\rangle$, $\langle$*free*, not *ai*, not *si*$\rangle$, $\langle$*free*, not *ai*, *si*$\rangle$, and $\langle$*free*, *ai*, *si*$\rangle$, respectively.

It is not difficult to check that action a is *si* but neither *free* nor *ai* in the synchronized automaton $\mathcal{T}$ of Example 4.2.1, depicted in Figure 4.6(b). This concludes our display of the remaining six combinations. □

We conclude by noting that the definitions of *free*, *ai*, and *si* synchronizations are based on the maximal interpretation adopted in Section 4.2. We will come back to this in Subsection 7.2.1, where we will reconsider *free*, *ai*, and *si* synchronizations in a context in which precise information on the participation of loops in synchronizations is available.

4.5 Predicates of Synchronizations

Our exposition until now has been analytical, in the sense that we have investigated transition relations to determine whether or not they satisfy the conditions inherent to certain types of synchronization. These conditions in general do not lead to uniquely defined synchronized automata.

In this section we deal with the question of how to describe a unique synchronized automaton, given a set of automata and certain conditions to be satisfied by the synchronizations. Recall that all elements of a synchronized automaton, except for its set of transitions, are uniquely determined by the set of automata it is composed over.

We begin by describing specific synchronized automata satisfying certain constraints on synchronizations. Synchronization constraints for an action a are conditions on the a-transitions to be chosen from $\Delta_a(\mathcal{S})$, the complete transition space of a in $\mathcal{S}$. Together, these conditions should determine a unique subset $\mathcal{R}_a$, which will be the set of a-transitions in the synchronized automaton. We will refer to subsets of the complete transition space $\Delta_a(\mathcal{S})$ as *predicates* (*of synchronizations*) for a. Once predicates have been chosen for all actions, the synchronized automaton over $\mathcal{S}$ defined by these predicates is unique.

The following generic definition formalizes this setup.

Definition 4.5.1. *For all $a \in \Sigma$, let $\mathcal{R}_a(\mathcal{S}) \subseteq \Delta_a(\mathcal{S})$ and let $\mathcal{R} = \{\mathcal{R}_a(\mathcal{S}) \mid a \in \Sigma\}$. Then $\mathcal{T}$ is* the $\mathcal{R}$-synchronized automaton *over $\mathcal{S}$ if for all $a \in \Sigma$,*

$$\delta_a = \mathcal{R}_a(\mathcal{S}). \qquad \square$$

A natural way of fixing a predicate for a given type of synchronization is to apply a maximality principle. Since a predicate is a subset of the complete transition space, this amounts to including everything that is not forbidden, i.e. everything that is in accordance with the chosen type of synchronization. This is the intuitive approach of [Ell97] and generalizes the classical approach to define synchronized systems from *ai* to other types of synchronization (cf. the Introduction). Thus when a synchronized automaton is to be constructed according to a specification of synchronization conditions for its set of actions, the strategy is to include as many transitions as possible without violating the specification, while checking that the result is unique.

This leads to the following predicates.

Definition 4.5.2. *Let $a \in \Sigma$. Then*

(1) the predicate no-constraints *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{no}(\mathcal{S})$ and is defined as*

$$\mathcal{R}_a^{no}(\mathcal{S}) = \Delta_a(\mathcal{S}),$$

(2) the predicate is-free *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{free}(\mathcal{S})$ and is defined as*

$$\mathcal{R}_a^{free}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \,|\, \#\{i \in \mathcal{I} \mid a \in \Sigma_i \wedge proj_i^{[2]}(q,q') \in \delta_{i,a}\} = 1\},$$

(3) the predicate is-ai *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{ai}(\mathcal{S})$ and is defined as*

$$\mathcal{R}_a^{ai}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid \forall i \in \mathcal{I} : a \in \Sigma_i \Rightarrow proj_i^{[2]}(q,q') \in \delta_{i,a}\}, \text{ and}$$

(4) the predicate is-si *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{si}(\mathcal{S})$ and is defined as*

$$\mathcal{R}_a^{si}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid \forall i \in \mathcal{I} : (a \in \Sigma_i \wedge a \ en_{\mathcal{A}_i} \ proj_i(q)) \Rightarrow proj_i^{[2]}(q,q') \in \delta_{i,a}\}. \ \square$$

Each of these predicates selects, for a given action a, *all* transitions from its complete transition space $\Delta_a(\mathcal{S})$ that obey a certain type of synchronization. In the case of *no-constraints* for a, this means that all a-transitions are allowed since nothing is required (and thus no transition is forbidden). In the other three cases, *all and only* those a-transitions are included that respect the specified property of a.

Theorem 4.5.3. *Let $a \in \Sigma$. Then*

(1) $a \in Free(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{free}(\mathcal{S})$,

(2) $a \in AI(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, and

(3) $a \in SI(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{si}(\mathcal{S})$.

Proof. Immediately from Definitions 4.4.1, 4.4.3, 4.4.5, and 4.5.2. □

The predicate $\mathcal{R}_a^{free}(\mathcal{S})$ ($\mathcal{R}_a^{ai}(\mathcal{S})$, $\mathcal{R}_a^{si}(\mathcal{S})$) thus defines the largest transition relation in $\Delta_a(\mathcal{S})$ in which an action a is *free* (*ai*, *si*). In other words, each of the types of synchronization introduced in the previous section gives rise to a predicate that is the unique maximal representative among all transition relations satisfying the type of synchronization.

Definition 4.5.4. *Let $syn \in \{free, ai, si\}$. Then*

(1) the $\{\mathcal{R}_a^{syn}(\mathcal{S}) \mid a \in \Sigma\}$-synchronized automaton over $\mathcal{S}$ is called the maximal-syn synchronized automaton *(over $\mathcal{S}$) and*

(2) an action $a \in \Sigma$ is called maximal-syn *in $\mathcal{T}$ if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$.* □

In case the automata from $\mathcal{S}$ have no shared actions, then the *maximal-free* (*maximal-ai*, *maximal-si*) synchronized automaton equals the $\mathcal{R}^{no}$-synchronized automaton (over $\mathcal{S}$).

Theorem 4.5.5. *Let $a \in \Sigma_j \setminus (\bigcup_{i \in \mathcal{I} \setminus \{j\}} \Sigma_i)$. Then*

$\mathcal{R}_a^{no}(\mathcal{S}) = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $syn \in \{free, ai, si\}$. □

4.6 Effect of Synchronizations

In this section we study the effect that the types of synchronization introduced in the previous sections have on the inheritance of the automata-theoretic properties from Section 3.2. We investigate both top-down inheritance — from synchronized automata to their (sub)automata — and bottom-up preservation — from (sub)automata to synchronized automata.

Notation 3. *For the remainder of this chapter we fix an arbitrary $j \in \mathcal{I}$ and an arbitrary subset $J \subseteq \mathcal{I}$. The subautomaton SUB_J of $\mathcal{T}$ will be specified as $SUB_J = (Q_J, \Sigma_J, \delta_J, I_J)$. We moreover fix Θ to be an arbitrary alphabet disjoint from Q.* □

The properties whose inheritance we study are static, in the sense that they depend on the mere "presence" of transitions in (sub)automata and synchronized automata. We begin by introducing two useful auxiliary notions.

A transition (p, a, p') of automaton $\mathcal{A}_j$ defines the execution of an action a by taking $\mathcal{A}_j$ from a (local) state p to a (local) state p'. Such a transition is *present* in the synchronized automaton $\mathcal{T}$ if it participates in one or more of the transitions of $\mathcal{T}$. In other words, if $\mathcal{T}$ can execute a by going from a (global) state q such that $\mathrm{proj}_j(q) = p$ to a (global) state q' such that $\mathrm{proj}_j(q') = p'$. The transition (p, a, p') is *omnipresent* in $\mathcal{T}$ if for all (global) states q of $\mathcal{T}$ such that $\mathrm{proj}_j(q) = p$, it can always be executed by participating in an a-transition (q, a, q') of $\mathcal{T}$ with $\mathrm{proj}_j(q') = p'$. The presence and omnipresence of transitions of SUB_J is defined likewise.

Definition 4.6.1. *(1) Let $(p, a, p') \in \delta_J$. Then*

- *(a) (p, a, p') is* present *in $\mathcal{T}$ if there exists a $(q, a, q') \in \delta$ such that $(proj_J(q), a, proj_J(q')) = (p, a, p')$ and*
- *(b) (p, a, p') is* omnipresent *in $\mathcal{T}$ if for all $q \in Q$ such that $proj_J(q) = p$, there exists a $(q, a, q') \in \delta$ such that $proj_J(q') = p'$.*

(2) Let $(p, a, p') \in \delta_j$. Then

- *(a) (p, a, p') is* present *in $\mathcal{T}$ if there exists a $(q, a, q') \in \delta$ such that $(proj_j(q), a, proj_j(q')) = (p, a, p')$ and*
- *(b) (p, a, p') is* omnipresent *in $\mathcal{T}$ if for all $q \in Q$ such that $proj_j(q) = p$, there exists a $(q, a, q') \in \delta$ such that $proj_j(q') = p'$.* □

Note that any transition of a (sub)automaton that is omnipresent in $\mathcal{T}$ is also present in $\mathcal{T}$.

We now investigate which conditions guarantee the presence or even omnipresence of the transitions of (sub)automata in synchronizations of synchronized automata over these (sub)automata. We are particularly interested in the presence or omnipresence of transitions in case of *free*, *ai*, and *si* actions.

As the transitions of any subautomaton of $\mathcal{T}$ are obtained from transitions of $\mathcal{T}$ by projection, each transition of a subautomaton of $\mathcal{T}$ is present — but not necessarily omnipresent — in $\mathcal{T}$.

Theorem 4.6.2. *Each transition of SUB_J is present in $\mathcal{T}$.* □

Since the transition relation of $\mathcal{T}$ is *chosen* from the complete transition space, certain transitions of automata from $\mathcal{S}$ may not be present (and thus neither omnipresent) in $\mathcal{T}$. We now study the types of synchronized automata in which not too many transitions from the complete transition space have been left out, i.e. in which transitions are (omni)present.

In the *maximal-si* synchronized automaton $\mathcal{T}$ over $\mathcal{S}$, all executions of an action a by definition involve all automata in which a is enabled at the current local state. Hence it is not surprising that all transitions of (sub)automata from $\mathcal{S}$ are omnipresent — and thus present — in $\mathcal{T}$.

Theorem 4.6.3. *Let $a \in \Sigma$.*

> *if $\delta_a = \mathcal{R}_a^{si}(\mathcal{S})$, then each a-transition of SUB_J as well as each a-transition of $\mathcal{A}_j$ is omnipresent in $\mathcal{T}$.*

Proof. We only prove the statement for SUB_J, as the other case is analogous. Let $\delta_a = \mathcal{R}_a^{si}(\mathcal{S})$ and let $(p, a, p') \in \delta_J$. Now let $q \in Q$ be such that $\mathrm{proj}_J(q) = p$ and let $q' \in Q$ be the state that is defined by $\mathrm{proj}_J(q') = p'$ and, for all $i \in \mathcal{I} \setminus J$, $\mathrm{proj}_i(q')$ is such that $(\mathrm{proj}_i(q), a, \mathrm{proj}_i(q')) \in \delta_i$ whenever $a \ \mathrm{en}_{\mathcal{A}_i} \ \mathrm{proj}_i(q)$. Then by Definitions 4.1.1 and 4.5.2(4), $(q, a, q') \in \mathcal{R}_a^{si}(\mathcal{S})$. Hence (p, a, p') is omnipresent in $\mathcal{T}$. □

It is clear that once a transition of an automaton is present or omnipresent in a synchronized automaton, adding more transitions to the latter will not affect that property. We may thus conclude from Theorem 4.6.3 that whenever $\mathcal{T}$ is such that $\delta_a = \mathcal{R}_a^{no}(\mathcal{S})$, for all $a \in \Sigma_{ext}$, then all transitions of the automata from $\mathcal{S}$ are omnipresent — and thus present — in $\mathcal{T}$. Moreover, if $\delta_a = \mathcal{R}_a^{no}(\mathcal{S})$, for all $a \in \Sigma_{ext}$, then for every transition (p, a, p') of SUB_J, we have that $(q, a, q') \in \mathcal{R}_a^{no}(\mathcal{S})$ for all $q \in Q$ such that $\mathrm{proj}_J(q) = p$, $\mathrm{proj}_J(q') = p'$, and for all $i \in \mathcal{I} \setminus J$, $\mathrm{proj}_i(q) = \mathrm{proj}_i(q')$.

Theorem 4.6.4. *Let $a \in \Sigma$. Then* *if* $\delta_a = \mathcal{R}_a^{no}(\mathcal{S})$*, then each* a*-transition of* SUB_J *as well as each* a*-transition of* $\mathcal{A}_j$ *is omnipresent in* $\mathcal{T}$*.* □

In the following example we demonstrate that in the *maximal-free* (*maximal-ai*) synchronized automaton over $\mathcal{S}$, not all transitions of all automata from $\mathcal{S}$ need to be present — let alone omnipresent. Apparently the *is-free* (*is-ai*) predicate may contain too few transitions from the complete transition space.

Example 4.6.5. Consider automata $\mathcal{A}_1 = (\{p\}, \{a\}, \{(p, a, p)\}, \{p\})$, $\mathcal{A}_2 = (\{q, q'\}, \{a\}, \{(q, a, q), (q, a, q'), (q', a, q')\}, \{q\})$, and $\mathcal{A}_3 = (\{r\}, \{a\}, \varnothing, \{r\})$. They are depicted in Figure 4.11.

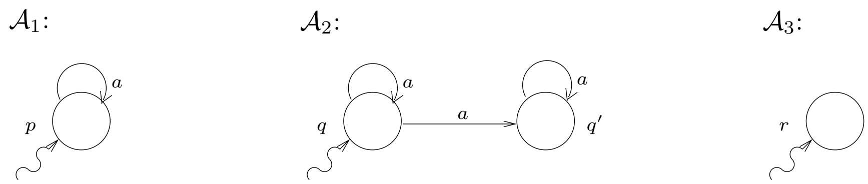


Fig. 4.11. Automata $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$.

It is not difficult to see that both the $\mathcal{R}^{free}$-synchronized automaton $\mathcal{T}_{1,2}^{free}$ over $\{\mathcal{A}_1, \mathcal{A}_2\}$ and the $\mathcal{R}^{ai}$-synchronized automaton $\mathcal{T}_{2,3}^{ai}$ over $\{\mathcal{A}_2, \mathcal{A}_3\}$ have an empty transition relation. We thus see that none of the a-transitions appearing in $\mathcal{A}_2$ is present — and thus neither omnipresent — in either $\mathcal{T}_{1,2}^{free}$ or $\mathcal{T}_{2,3}^{ai}$. □

By looking more closely at Example 4.6.5 we obtain some hints as to why some transitions of automata from $\mathcal{S}$ cannot be omnipresent in the *maximal-free* (*maximal-ai*) synchronized automaton over $\mathcal{S}$.

First consider the case that $\mathcal{T}$ is the *maximal-ai* synchronized automaton over $\mathcal{S}$. From Example 4.6.5 it follows immediately that no a-transition of $\mathcal{A}_j$ will be present in $\mathcal{T}$ if $\delta_a = \varnothing$. On the other hand, if $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \neq \varnothing$, then every a-transition of $\mathcal{A}_j$ can be executed in $\mathcal{T}$ from every state in which a is enabled at the local states of all other automata that also have a as an action.

Theorem 4.6.6. *For all* $a \in \Theta \cap \Sigma_j$*, let* $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$*. Then*

$\mathcal{R}_a^{ai}(\mathcal{S}) \neq \varnothing$ *if and only if* $\delta_{j,a} \neq \varnothing$ *and each* a*-transition of* $\mathcal{A}_j$ *is present in* $\mathcal{T}$*.*

Proof. (If) Trivial.

(Only if) Let $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \neq \varnothing$. Then for all $i \in \mathcal{I}$, if $a \in \Sigma_i$, then there exist q_i, q_i' such that $(q_i, a, q_i') \in \delta_i$. Now let $(p, a, p') \in \delta_j$ and let $q, q' \in Q$ be such that $\mathrm{proj}_j(q) = p$ and $\mathrm{proj}_j(q') = p'$, $\mathrm{proj}_i(q) = q_i$ and $\mathrm{proj}_i(q') = q_i'$, for all $i \in \mathcal{I}$ such that $a \in \Sigma_i$ and $i \neq j$, and $\mathrm{proj}_k(q) = \mathrm{proj}_k(q')$, for all $k \in \mathcal{I}$ such that $a \notin \Sigma_k$. This implies that $(q, a, q') \in \mathcal{R}_a^{ai}(\mathcal{S})$ and hence (p, a, p') is present in $\mathcal{T}$. □

Example 4.6.5 suggests furthermore that certain transitions of automata from $\mathcal{S}$ cannot be omnipresent in $\mathcal{T}$ in case the following situation exists. Let q be a state of $\mathcal{T}$ at which an action a is locally enabled — due to the existence of an a-transition t — in (at least) one of the automata from $\mathcal{S}$, while it is not locally enabled — due to the absence of an a-transition — in (at least) one other automaton from $\mathcal{S}$ *that does have a in its alphabet*. If this is the case, then a is not enabled at q in $\mathcal{T}$. The reason is that otherwise action a could be executed from q *without* the participation of all of the automata having this a as one of their actions, which would be contradicting the fact that $\mathcal{T}$ is the *maximal-ai* synchronized automaton over $\mathcal{S}$. Hence the a-transition t cannot be omnipresent in $\mathcal{T}$.

To avoid the situation sketched above from occurring when dealing with *maximal-ai* synchronized automata, we define a *Θ-enabling set of automata* as a set of automata with the property that each of its constituting automata is Θ-enabling. Recall Θ to be an arbitrary alphabet disjoint from Q.

Definition 4.6.7. *$\mathcal{S}$ is Θ-enabling if for all $i \in \mathcal{I}$, $\mathcal{A}_i$ is Θ-enabling.* □

If $\mathcal{S}$ is Σ-enabling, then we may also simply say that $\mathcal{S}$ is enabling. Note, however, that in that case the *maximal-ai* synchronized automaton over $\mathcal{S}$ and the *maximal-si* synchronized automaton over $\mathcal{S}$ are one and the same. In fact, if $\mathcal{S}$ is $\{a\}$-enabling and $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$, for an action a, then clearly $\delta_a = \mathcal{R}_a^{si}(\mathcal{S})$.

Theorem 4.6.8. *For all $a \in \Theta \cap \Sigma$, let $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$. Then*

if $\mathcal{S}$ is Θ-enabling, then for all $a \in \Theta$, each a-transition of $SUB_{\mathcal{J}}$ as well as each a-transition of $\mathcal{A}_j$ is omnipresent in $\mathcal{T}$.

Proof. Let $\mathcal{S}$ be Θ-enabling. Together with the fact that $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma$, this implies that $\delta_a = \mathcal{R}_a^{si}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma$, after which the result follows directly from Theorem 4.6.3. □

We conclude that whenever $\mathcal{S}$ is enabling, all transitions of (sub)automata from $\mathcal{S}$ are omnipresent — and thus present — in the *maximal-ai* synchronized automaton $\mathcal{T}$ over $\mathcal{S}$.

Now consider the case that $\mathcal{T}$ is the *maximal-free* synchronized automaton over $\mathcal{S}$. Consequently, Example 4.6.5 suggests that certain transitions of automata from $\mathcal{S}$ cannot be omnipresent in $\mathcal{T}$ in case the following situation exists. Let q be a state of $\mathcal{T}$ at which an action a is locally enabled in (at least) two of the automata from $\mathcal{S}$, of which (at least) one time as a loop. Then the other a-transition that is locally enabled at q cannot be omnipresent in $\mathcal{T}$. The reason is that by our maximal interpretation the automaton with the loop on a participates in the execution of any a-transition in $\mathcal{T}$ from q. This would be contradicting the fact that $\mathcal{T}$ is the *maximal-free* synchronized automaton over $\mathcal{S}$.

To avoid the situation sketched above from occurring when studying *maximal-free* synchronized automata, we define a *Θ-J-loop-limited set of automata* as a set of automata with the property that whenever there is an a-transition, with $a \in \Theta$, in the *maximal-free* team automaton over $\mathcal{A}_k$, $k \in J$, then none of the other automata in the set has a loop on a.

Definition 4.6.9. *(1) $\mathcal{S}$ is Θ-J-*loop limited *if for all $a \in \Theta \cap \Sigma_J$, whenever there exists an $i \in \mathcal{I} \setminus J$ such that $(q,q) \in \delta_{i,a}$ for some $q \in Q_i$, then $\mathcal{R}_a^{free}(\{\mathcal{A}_k \mid k \in J\}) = \varnothing$, and*

*(2) $\mathcal{S}$ is Θ-j-*loop limited *if for all $a \in \Theta \cap \Sigma_j$, whenever there exists an $i \in \mathcal{I} \setminus \{j\}$ such that $(q,q) \in \delta_{i,a}$ for some $q \in Q_i$, then $\delta_{j,a} = \varnothing$.* □

We thus note that $\mathcal{S}$ being Θ-j-loop limited is the same as $\mathcal{S}$ being Θ-$\{j\}$-loop limited. If $\mathcal{S}$ is Σ_J-J-loop limited or Σ_j-j-loop limited, then we may also simply say that $\mathcal{S}$ is J-loop limited or j-loop limited, respectively. Finally, note that whenever $\Theta \subseteq \Sigma_J \setminus (\bigcup_{i \in \mathcal{I} \setminus J} \Sigma_i)$ or $\Theta \subseteq \Sigma_j \setminus (\bigcup_{i \in \mathcal{I} \setminus \{j\}} \Sigma_i)$, then $\mathcal{S}$ is Θ-J-loop limited or Θ-j-loop limited, respectively.

Loop limitedness is a sufficient *and* necessary condition on $\mathcal{S}$ for guaranteeing all transitions of (sub)automata from $\mathcal{S}$ to be omnipresent — and thus present — in the *maximal-free* synchronized automaton $\mathcal{T}$ over $\mathcal{S}$.

Theorem 4.6.10. *For all $a \in \Theta \cap \Sigma$, let $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$. Then*

(1) each a-transition of SUB_J, for all $a \in \Theta$, is omnipresent in $\mathcal{T}$ if and only if $\mathcal{S}$ is Θ-J-loop limited, and

(2) each a-transition of $\mathcal{A}_j$, for all $a \in \Theta$, is omnipresent in $\mathcal{T}$ if and only if $\mathcal{S}$ is Θ-j-loop limited.

Proof. (1) (If) Let $\mathcal{S}$ be Θ-J-loop limited, let $a \in \Theta$, and let $(p,a,p') \in \delta_J$. Now let $q \in Q$ be such that $\mathrm{proj}_J(q) = p$ and let $q' \in Q$ be the state that is defined by $\mathrm{proj}_J(q') = p'$ and, for all $i \in \mathcal{I} \setminus J$, $\mathrm{proj}_i(q') = \mathrm{proj}_i(q)$.

Then Definitions 4.1.1 and 4.5.2(2) together with the fact that $\mathcal{S}$ is Θ-J-loop limited imply that $(q, a, q') \in \mathcal{R}_a^{free}(\mathcal{S})$. Hence (p, a, p') is omnipresent in $\mathcal{T}$.

(Only if) Let each a-transition of SUB_J, for all $a \in \Theta$, be omnipresent in $\mathcal{T}$. Now assume that $\mathcal{S}$ is not Θ-J-loop limited. Then there exist an $a \in \Theta$, a $(p, a, p') \in \mathcal{R}_a^{free}(\{\mathcal{A}_k \mid k \in J\})$, and an $i \in \mathcal{I} \setminus J$ such that $(q, a, q) \in \delta_i$. Now let $r \in Q$ be such that $\mathrm{proj}_J(r) = p$ and $\mathrm{proj}_i(r) = q$. Since (p, a, p') is omnipresent in $\mathcal{T}$, there exists an $(r, a, r') \in \delta$ such that $\mathrm{proj}_J(r') = p'$. Moreover, because $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$ it must be the case that for all $\ell \in \mathcal{I} \setminus J$, $\mathrm{proj}_\ell(r') = \mathrm{proj}_\ell(r)$ and $(\mathrm{proj}_\ell(r), a, \mathrm{proj}_\ell(r')) \notin \delta_\ell$, which contradicts the fact that $(q, a, q) \in \delta_i$. Hence $\mathcal{S}$ is Θ-J-loop limited.

(2) Analogous. □

This concludes our intermezzo on the presence and omnipresence of transitions of (sub)automata in synchronized automata over these (sub)automata. In the next two subsections we investigate the inheritance of the automata-theoretic properties introduced in Section 3.2 from synchronized automata to their (sub)automata, and vice versa. While doing so we adhere to the order according to which these properties were introduced.

4.6.1 Top-Down Inheritance of Properties

Initially we search for sufficient conditions under which the automata-theoretic properties of Section 3.2 are inherited from synchronized automata to their (sub)automata.

Reduced Versions

In order to investigate the conditions under which action reducedness, transition reducedness, and state reducedness are inherited from a synchronized automaton to its (sub)automata, it is important to know whether or not the projection on a (sub)automaton of a state that is reachable in a synchronized automaton is itself reachable in that (sub)automaton.

Lemma 4.6.11. *Let $q \in Q$ be reachable in $\mathcal{T}$. Then*

(1) $\mathrm{proj}_J(q)$ is reachable in SUB_J and

(2) $\mathrm{proj}_j(q)$ is reachable in $\mathcal{A}_j$.

Proof. If $q \in Q$ is reachable in $\mathcal{T}$, then there exists a computation $\alpha q \in \mathbf{C}_\mathcal{T}$. Hence (1) and (2) follow directly from Lemma 4.2.6 and its Corollary 4.2.7, respectively. □

An immediate consequence of Lemma 4.6.11 is that the state reducedness of a synchronized automaton is inherited by all its (sub)automata.

Theorem 4.6.12. *Let $\mathcal{T}$ be state reduced. Then*

SUB_J as well as $\mathcal{A}_j$ is state reduced. □

Note that the statements of Lemma 4.6.11 cannot be reversed. This follows from Example 4.2.8. This also means that the Θ-action-reduced (Θ-transition-reduced) versions of the subautomata of a synchronized automaton in general are different from the subautomata of the Θ-action-reduced (Θ-transition-reduced) versions of that synchronized automaton. Hence in general $SUB_J(\mathcal{T}_A^\Theta) \neq (SUB_J(\mathcal{T}))_A^\Theta$ and $SUB_J(\mathcal{T}_T^\Theta) \neq (SUB_J(\mathcal{T}))_T^\Theta$, even if $\Theta \subseteq \Sigma_J$. The situation is different in case of state reducedness. In fact, since the state-reduced version $\mathcal{T}_S$ of a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$ need not be a synchronized automaton over $\mathcal{S}$, subautomata of $\mathcal{T}_S$ are not defined unless $\mathcal{T}_S = \mathcal{T}$, i.e. $\mathcal{T}$ is state reduced. However, if $\mathcal{T}$ is state reduced, then Theorem 4.6.12 implies that $SUB_J(\mathcal{T}) = (SUB_J(\mathcal{T}))_S$.

In the following example we show that the fact that $\mathcal{T}$ is Θ-action reduced (Θ-transition reduced) in general does not imply that each of its constituting automata is Θ-action reduced (Θ-transition reduced). To construct a Θ-action-reduced synchronized automaton $\mathcal{T}$ over $\mathcal{S}$, it suffices to have just one Θ-action-reduced automaton in $\mathcal{S}$. By basing the transition relation of $\mathcal{T}$ solely on that Θ-action-reduced automaton, e.g., one obtains that $\mathcal{T}$ is Θ-action reduced while obviously not all automata from $\mathcal{S}$ need to be Θ-action reduced. It is even easier to construct a Θ-transition-reduced synchronized automaton $\mathcal{T}$ over $\mathcal{S}$, viz. by equipping $\mathcal{T}$ with only useful a-transitions, for all $a \in \Theta$.

Example 4.6.13. Consider automata $\mathcal{A}_1 = (\{q_1, q_1'\}, \{a\}, \{(q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{A}_2 = (\{q_2, q_2'\}, \{a\}, \{(q_2', a, q_2)\}, \{q_2\})$, as depicted in Figure 4.12(a).

Consider the synchronized automaton $\mathcal{T} = (Q, \{a\}, \delta, \{(q_1, q_2)\})$, with $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$ and $\delta = \{((q_1, q_2), a, (q_1', q_2))\}$, over $\{\mathcal{A}_1, \mathcal{A}_2\}$. It is depicted in Figure 4.12(b).

It is easy to see that $\mathcal{T}$ is both action reduced and transition reduced, whereas $\mathcal{A}_2$ clearly is neither action reduced nor transition reduced. □

The action reducedness of a synchronized automaton *is* inherited by each of its (sub)automata in case each of the latter's actions is *ai* in the synchronized automaton.

Theorem 4.6.14. *Let $\mathcal{T}$ be Θ-action reduced. Then*

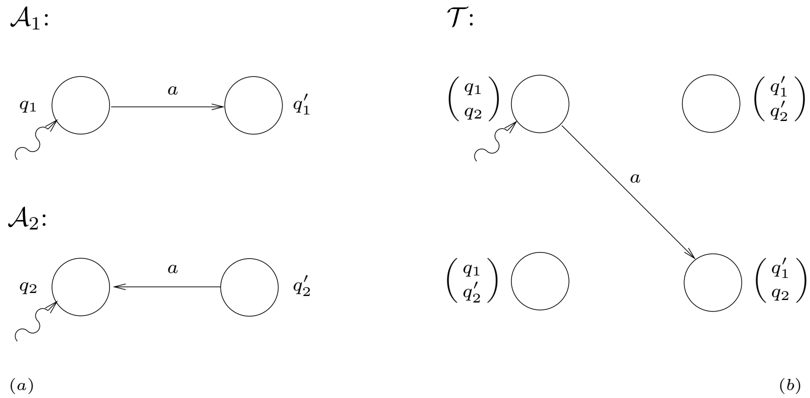


Fig. 4.12. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$, and synchronized automaton $\mathcal{T}$.

(1) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_J$, then SUB_J is Θ-action reduced, and

(2) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{A}_j$ is Θ-action reduced.

Proof. (1) Let $a \in \Theta \cap \Sigma_J$ and let $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$. Since $\mathcal{T}$ is Θ-action reduced we know that there exists a computation $\alpha \in \mathbf{C}_{\mathcal{T}}$ such that $\alpha = \beta a q$ for some $\beta \in I(\Sigma Q)^*$ and $q \in Q$. Since $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, $\pi_{SUB_J}(\alpha) = \pi_{SUB_J}(\beta) a \mathrm{proj}_J(q) \in \mathbf{C}_{SUB_J}$ by Definition 4.2.3(1) and Lemma 4.2.6. Hence a is active in SUB_J and SUB_J is thus Θ-action reduced.

(2) Analogous, but now using Definition 4.2.3(3) and Corollary 4.2.7. □

It is worthwhile to notice that the requirement of every action being *ai* as condition in this theorem cannot be replaced by requiring each action to be *free* or *si* without invalidating the statement. In the following example we show this by demonstrating that the action reducedness of $\mathcal{T}$ in general is not inherited by each of its (sub)automata in case $\mathcal{T}$ is the *maximal-free* synchronized automaton nor in case $\mathcal{T}$ is the *maximal-si* synchronized automaton — and hence neither in case $\mathcal{T}$ is a synchronized automaton in which every action is *free* nor in case $\mathcal{T}$ is a synchronized automaton in which every action is *si*.

Example 4.6.15. (Example 4.6.13 continued) First we consider the $\mathcal{R}^{free}$-synchronized automaton $\mathcal{T}^{free} = (Q, \{a\}, \delta^{free}, \{(q_1, q_2)\})$, with $\delta^{free} = \delta \cup \{((q_1, q_2'), a, (q_1, q_2)), ((q_1, q_2'), a, (q_1', q_2')), ((q_1', q_2'), a, (q_1', q_2))\}$, over $\{\mathcal{A}_1, \mathcal{A}_2\}$. It is depicted in Figure 4.13(a).

Clearly, $\mathcal{T}^{free}$ is $\{a\}$-action reduced. Now note that $SUB_{\{2\}}(\mathcal{T}^{free})$ is essentially a copy of $\mathcal{A}_2$. It is easy to see that neither $SUB_{\{2\}}(\mathcal{T}^{free})$ nor $\mathcal{A}_2$ is $\{a\}$-action reduced.

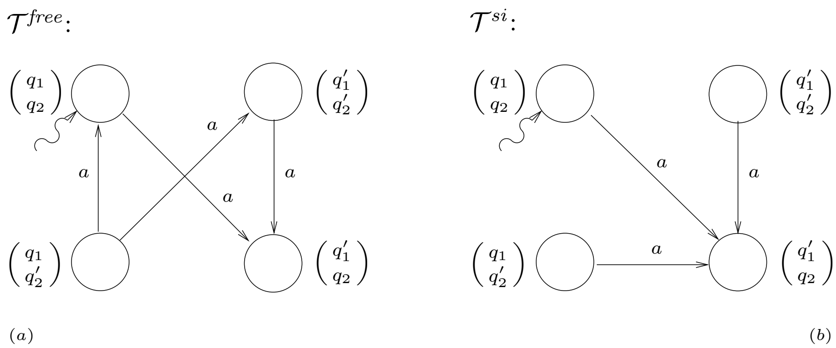


Fig. 4.13. Synchronized automata $\mathcal{T}^{free}$ and $\mathcal{T}^{si}$.

Next we consider the $\mathcal{R}^{si}$-synchronized automaton $\mathcal{T}^{si} = (Q, \{a\}, \delta^{si}, \{(q_1, q_2)\})$, with $\delta^{si} = \delta \cup \{((q_1, q_2'), a, (q_1', q_2)), ((q_1', q_2'), a, (q_1', q_2))\}$, over $\{\mathcal{A}_1, \mathcal{A}_2\}$. It is depicted in Figure 4.13(b).

Clearly, $\mathcal{T}^{si}$ is $\{a\}$-action reduced. Moreover, also $SUB_{\{2\}}(\mathcal{T}^{si})$ is essentially a copy of $\mathcal{A}_2$. Since we know that $\mathcal{A}_2$ is not $\{a\}$-action reduced, neither is $SUB_{\{2\}}(\mathcal{T}^{si})$.

Finally, we note that the $\mathcal{R}^{ai}$-synchronized automaton $\mathcal{T}^{ai} = (Q, \{a\}, \{((q_1, q_2'), a, (q_1', q_2))\}, \{(q_1, q_2)\})$ over $\{\mathcal{A}_1, \mathcal{A}_2\}$ is not $\{a\}$-action reduced. □

In Example 4.6.13 we have seen that the fact that $\mathcal{T}$ is transition reduced in general does not imply that $\mathcal{A}_j$ is transition reduced. As we show next, the transition reducedness of a synchronized automaton *is* inherited by each of its (sub)automata in case each of the latter's transitions is present in the synchronized automaton.

Theorem 4.6.16. *Let $\mathcal{T}$ be Θ-transition reduced. Then*

(1) SUB_J is Θ-transition reduced and

(2) if each a-transition of $\mathcal{A}_j$, for all $a \in \Theta$, is present in $\mathcal{T}$, then $\mathcal{A}_j$ is Θ-transition reduced.

Proof. (1) Let $a \in \Theta \cap \Sigma_J$ and let $(p, a, p') \in \delta_J$. Then Theorem 4.6.2 implies that there exists a transition $(q, a, q') \in \delta$ such that $(\mathrm{proj}_J(q), a, \mathrm{proj}_J(q')) = (p, a, p')$. Since $\mathcal{T}$ is Θ-transition reduced there furthermore exists a computation $\alpha q \in \mathbf{C}_{\mathcal{T}}$, i.e. q is reachable in $\mathcal{T}$. Lemma 4.6.11(1) now implies that p is reachable in SUB_J and thus (p, a, p') is useful in SUB_J. Hence SUB_J is Θ-transition reduced.

(2) Analogous. □

Together with Theorems 4.6.3, 4.6.4, 4.6.6, and 4.6.10(2) this implies the following result.

Corollary 4.6.17. *Let $\mathcal{T}$ be Θ-transition reduced and let syn $\in \{si, no\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{A}_j$ is Θ-transition reduced,

(2) if $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \neq \varnothing$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{A}_j$ is Θ-transition reduced, and

(3) if $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, and $\mathcal{S}$ is Θ-j-loop limited, then $\mathcal{A}_j$ is Θ-transition reduced. □

Enabling

We now turn to the inheritance of enabling from synchronized automata to their (sub)automata. In the following example we show that when a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$ is Θ-enabling, then this in general does not imply that each of its (sub)automata is Θ-enabling. We show this by using the fact that a necessary condition for a synchronized automaton to be $\{a\}$-enabling, for an action a, is that in each of its states (at least) one of its constituting automata enables a. However, it is not guaranteed that each of the synchronized automaton's (sub)automata enables a in each of its states.

Example 4.6.18. (Example 4.2.1 continued) Clearly $\mathcal{T}$ is action reduced and state reduced (and thus transition reduced). It is moreover enabling. However, we immediately see that $\mathcal{A}_1$ and $\mathcal{A}_3$ are not. It is also easy to see that $SUB_{\{3\}}$, which is essentially a copy of $\mathcal{A}_3$, is not enabling. □

Note that this example allows us to conclude that also the Θ-enabling of a Θ-action-reduced (Θ-transition-reduced, state-reduced) synchronized automaton in general is not inherited by its (sub)automata.

The enabling of a synchronized automaton *is* inherited by each of its (sub)automata in case every action of the synchronized automaton is *ai*.

Theorem 4.6.19. *Let $\mathcal{T}$ be Θ-enabling. Then*

(1) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_J$, then SUB_J is Θ-enabling, and

(2) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{A}_j$ is Θ-enabling.

Proof. (1) Let $a \in \Theta \cap \Sigma_J$ and let $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$. Let $p \in Q_J$. Now let $q \in Q$ be such that $\mathrm{proj}_J(q) = p$. Since $\mathcal{T}$ is Θ-enabling we know that $a \ \mathrm{en}_{\mathcal{T}} \ q$. Hence there exists a $q' \in Q$ such that $(q, q') \in \delta_a$. Moreover, $\mathrm{proj}_J{}^{[2]}(q, q') \in (\delta_J)_a$ because $a \in \Sigma_J$ and $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$. Consequently, $a \ \mathrm{en}_{SUB_J} \ p$. Hence SUB_J is Θ-enabling.

(2) Analogous. □

It is worthwhile to notice that the requirement of every action being *ai* as condition in this theorem cannot be replaced by requiring each action to be *free* or *si* without invalidating the statement. In the following example we show this by demonstrating that the enabling of $\mathcal{T}$ in general is not inherited by each of its (sub)automata in case $\mathcal{T}$ is the *maximal-free* synchronized automaton nor in case $\mathcal{T}$ is the *maximal-si* synchronized automaton — and hence neither in case $\mathcal{T}$ is a synchronized automaton in which every action is *free* nor in case $\mathcal{T}$ is a synchronized automaton in which every action is *si*.

Example 4.6.20. Let $\mathcal{A}_1 = (\{q_1, q_1'\}, \{a\}, \{(q_1, a, q_1'), (q_1', a, q_1)\}, \{q_1\})$ and let $\mathcal{A}_2 = (\{q_2, q_2'\}, \{a\}, \{(q_2, a, q_2')\}, \{q_2\})$. These automata are depicted in Figure 4.14.

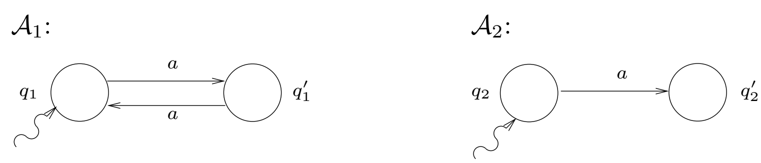


Fig. 4.14. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

In Figure 4.15(a) we have depicted the $\mathcal{R}^{free}$-synchronized automaton $\mathcal{T}^{free} = (Q, \{a\}, \delta^{free}, \{(q_1, q_2)\})$, with $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$ and δ^{free} as depicted, over $\{\mathcal{A}_1, \mathcal{A}_2\}$.

It is easy to see that $\mathcal{T}^{free}$ is enabling. Now note that $SUB_{\{2\}}(\mathcal{T}^{free})$ is essentially a copy of $\mathcal{A}_2$. Clearly neither $SUB_{\{2\}}(\mathcal{T}^{free})$ nor $\mathcal{A}_2$ is enabling.

Consequently, in Figure 4.15(b) we have depicted the $\mathcal{R}^{si}$-synchronized automaton $\mathcal{T}^{si} = (Q, \{a\}, \delta^{si}, \{(q_1, q_2)\})$, with δ^{si} as depicted, over $\{\mathcal{A}_1, \mathcal{A}_2\}$.

It is again easy to see that $\mathcal{T}^{si}$ is enabling. Clearly also $SUB_{\{2\}}(\mathcal{T}^{si})$ is essentially a copy of $\mathcal{A}_2$. Since $\mathcal{A}_2$ is not enabling, neither is $SUB_{\{2\}}(\mathcal{T}^{si})$.

Finally, we note that the $\mathcal{R}^{ai}$-synchronized automaton $\mathcal{T}^{ai} = (Q, \{a\}, \{((q_1, q_2), a, (q_1', q_2')), ((q_1', q_2), a, (q_1, q_2'))\}, \{(q_1, q_2)\})$ over $\{\mathcal{A}_1, \mathcal{A}_2\}$ is not enabling. □

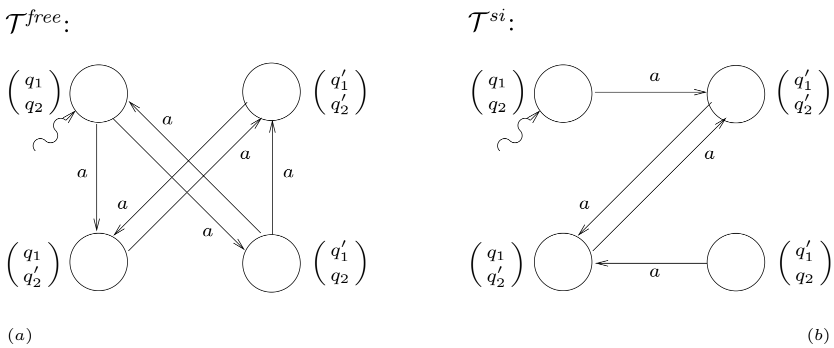


Fig. 4.15. Synchronized automata $\mathcal{T}^{free}$ and $\mathcal{T}^{si}$.

Determinism

We now conclude this subsection by turning to the inheritance of determinism. We begin by showing that when a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$ is Θ-deterministic, then this in general does not imply that each of its (sub)automata is Θ-deterministic. In case of inheritance from a synchronized automaton to its constituting automata, this can be concluded directly from Example 4.6.5. In case of inheritance from a synchronized automaton to its subautomata, this can be concluded from the following example. This example uses the fact that the states of $\mathcal{A}_j$ can be used to distinguish states of a synchronized automaton $\mathcal{T}$ that without the j-th component cannot be distinguished.

Example 4.6.21. Consider automata $\mathcal{A}_1 = (\{p,p'\},\{a\},\{(p,a,p'),(p',a,p)\},\{p\})$, $\mathcal{A}_2 = (\{q,q'\},\{a\},\{(q,a,q'),(q',a,q)\},\{q\})$, and $\mathcal{A}_3 = (\{r,r'\},\{a\},\{(r,a,r'),(r',a,r)\},\{r\})$, as depicted in Figure 4.16.

In Figure 4.17(a) we have depicted the synchronized automaton $\mathcal{T} = (Q,\{a\},\delta,\{(p,q,r)\})$, with $Q = \{(p,q,r),(p',q,r),(p,q',r),(p',q',r),(p,q,r'),(p',q,r'),(p,q',r'),(p',q',r')\}$ and δ as depicted, over $\{\mathcal{A}_i \mid i \in [3]\}$.

It is easy to see that $\mathcal{T}$ is action reduced and state reduced (and thus transition reduced). Furthermore, $\mathcal{T}$ clearly is deterministic.

Consequently, in Figure 4.17(b) we have depicted its subautomaton $SUB_{\{1,2\}} = (\{(p,q),(p,q'),(p',q),(p',q')\},\{a\},\delta_{\{1,2\}},\{(p,q)\})$, with $\delta_{\{1,2\}}$ as depicted.

Clearly $SUB_{\{1,2\}}$ is not deterministic as, e.g., $((p',q),a,(p,q)) \in \delta_{\{1,2\}}$ and $((p',q),a,(p,q')) \in \delta_{\{1,2\}}$. □

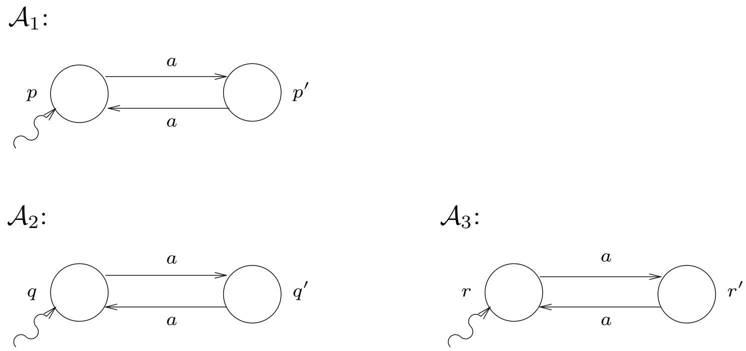


Fig. 4.16. Automata $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$.

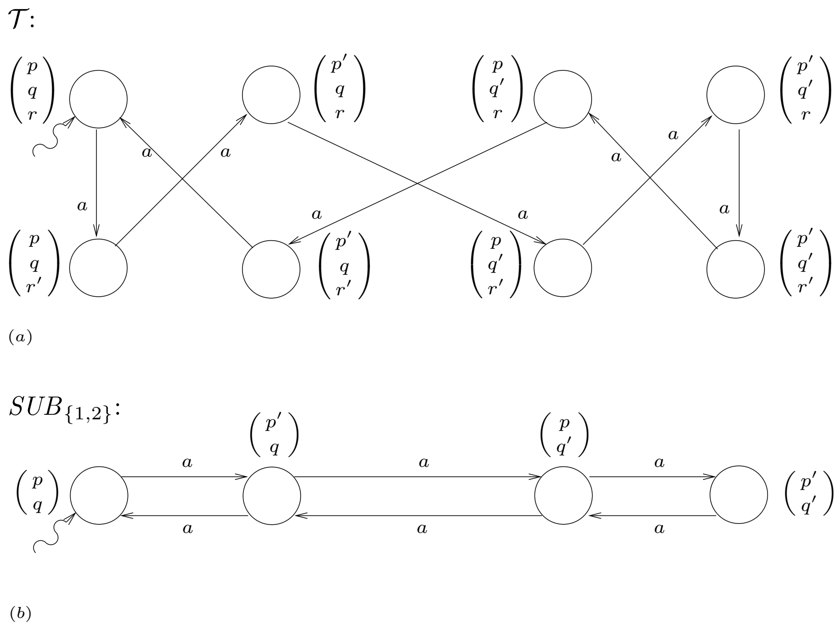


Fig. 4.17. Synchronized automaton $\mathcal{T}$ and its subautomaton $SUB_{\{1,2\}}$.

The determinism of a *maximal-free* (*maximal-ai*, *maximal-si*) synchronized automaton *is* inherited by each of its (sub)automata in case each of the latter's transitions is present in the synchronized automaton.

Theorem 4.6.22. *Let $\mathcal{T}$ be Θ-deterministic and let $syn \in \{no, free, ai, si\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_J$, then SUB_J is Θ-deterministic, and

(2) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$ and each a-transition of $\mathcal{A}_j$ is present in $\mathcal{T}$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{A}_j$ is Θ-deterministic.

Proof. (1) Let $a \in \Theta \cap \Sigma_J$ and let $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$. Since $\mathcal{T}$ is Θ-deterministic we know that $I = \{q_0\}$, for some $q_0 \in Q$. Hence, trivially, $I_J = \{\mathrm{proj}_J(q_0)\}$. It thus remains to prove that for all $q \in Q_J$, there exists at most one $q' \in Q_J$ such that $(q, a, q') \in \delta_J$.

Now assume that there exists a $p \in Q_J$ such that $(p, a, p') \in \delta_J$ and $(p, a, p'') \in \delta_J$, with $p' \neq p''$. Then Theorem 4.6.2 implies that there exist a $(q, a, q') \in \delta$ such that $(\mathrm{proj}_J(q), a, \mathrm{proj}_J(q')) = (p, a, p')$ and an $(r, a, r') \in \delta$ such that $(\mathrm{proj}_J(r), a, \mathrm{proj}_J(r')) = (p, a, p'')$. Moreover, since $q' \neq r'$ and $\mathcal{T}$ is Θ-deterministic, we know that $q \neq r$. Consequently, the fact that $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$ implies that we can replace the components from J in (q, q') by those in (r, r') and still have a transition in $\mathcal{R}_a^{syn}(\mathcal{S})$. Hence there exists a $q'' \in Q$ such that $(q, a, q'') \in \delta$ with $\mathrm{proj}_{\mathcal{I} \setminus J}(q'') = \mathrm{proj}_{\mathcal{I} \setminus J}(q')$ and $\mathrm{proj}_J(q'') = p'' = \mathrm{proj}_J(r')$. Since $p' \neq p''$ this means that $\mathcal{T}$ is not Θ-deterministic, a contradiction. Hence SUB_J is Θ-deterministic.

(2) Analogous. □

Together with Theorems 4.6.3, 4.6.4, 4.6.6, and 4.6.10(2) this implies the following result.

Corollary 4.6.23. *Let $\mathcal{T}$ be Θ-deterministic and let $syn \in \{si, no\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{A}_j$ is Θ-deterministic,

(2) if $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \neq \varnothing$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{A}_j$ is Θ-deterministic, and

(3) if $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, and $\mathcal{S}$ is Θ-j-loop limited, then $\mathcal{A}_j$ is Θ-deterministic. □

4.6.2 Bottom-Up Inheritance of Properties

Dual to the above investigations we now change focus and study sufficient conditions under which the automata-theoretic properties of Section 3.2 are preserved from automata to synchronized automata.

We recall from Section 4.3 that $\mathcal{T}$ is a synchronized automaton over $\mathcal{S}'$ — upto a reordering — whenever $\mathcal{S}' = \{SUB_{\mathcal{I}_j} \mid \{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}\}$. Hence it suffices to investigate the conditions under which a property that holds for (elements of) a set of automata is preserved by a synchronized automaton over that set of automata. Therefore, we extend Definition 4.6.7 by defining when a set of automata is Θ-action reduced (Θ-transition reduced, state reduced, Θ-deterministic).

Definition 4.6.24. *$\mathcal{S}$ is Θ-action reduced (Θ-transition reduced, state reduced, Θ-deterministic) if for all $i \in \mathcal{I}$, $\mathcal{A}_i$ is Θ-action reduced (Θ-transition reduced, state reduced, Θ-deterministic).* □

If $\mathcal{S}$ is Σ-action reduced (Σ-transition reduced, Σ-deterministic) we may also simply say that $\mathcal{S}$ is action reduced (transition reduced, deterministic).

In the following example we show that the fact that $\mathcal{S}$ is Θ-action reduced (Θ-transition reduced, state reduced) in general does not imply that $\mathcal{T}$ is Θ-action reduced (Θ-transition reduced, state reduced). We moreover show that in case $\mathcal{S}$ is Θ-enabling (Θ-deterministic), then this in general does not imply that $\mathcal{T}$ is Θ-enabling (Θ-deterministic). To show this we use the fact that the transition relation of a synchronized automaton over a set of automata is chosen from the complete transition space. Hence we simply consider a set of automata that satisfies a certain property (i.e. each of its constituting automata satisfies this particular property) and consequently we choose the transition relation of a synchronized automaton over it in such a way that the property fails to hold for that particular synchronized automaton.

Example 4.6.25. Let automata $\mathcal{A}_1 = (\{q_1, q_1'\}, \{a\}, \{(q_1, a, q_1'), (q_1', a, q_1)\}, \{q_1\})$ and $\mathcal{A}_2 = (\{q_2, q_2'\}, \{a, b\}, \{(q_2, b, q_2), (q_2, a, q_2'), (q_2', b, q_2')\}, \{q_2\})$ be as depicted in Figure 4.18.

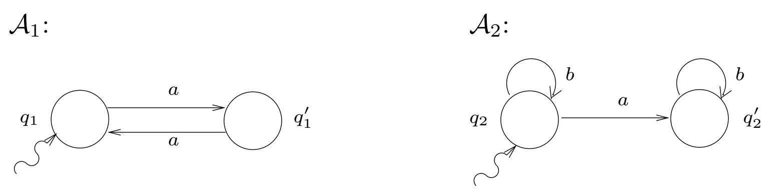


Fig. 4.18. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

It is easy to see that both $\mathcal{A}_1$ and $\mathcal{A}_2$ are action reduced, state reduced (and thus transition reduced), and deterministic. Moreover, $\mathcal{A}_1$ is enabling and $\mathcal{A}_2$ is $\{b\}$-enabling.

Now consider the synchronized automaton $\mathcal{T} = (Q, \{a, b\}, \delta, \{(q_1, q_2)\})$, where $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$ and $\delta = \{((q_1, q_2), a, (q_1', q_2')), ((q_1, q_2), a, (q_1, q_2')), ((q_1', q_2), a, (q_1, q_2'))\}$, over $\{\mathcal{A}_1, \mathcal{A}_2\}$. It is depicted in Figure 4.19(a).

Since b is not active in $\mathcal{T}$ it is clear that $\mathcal{T}$ is not action reduced. Furthermore, $\mathcal{T}$ is not transition reduced (and thus neither state reduced) since $((q_1', q_2), a, (q_1, q_2'))$ is not useful in $\mathcal{T}$. By removing both this transition and

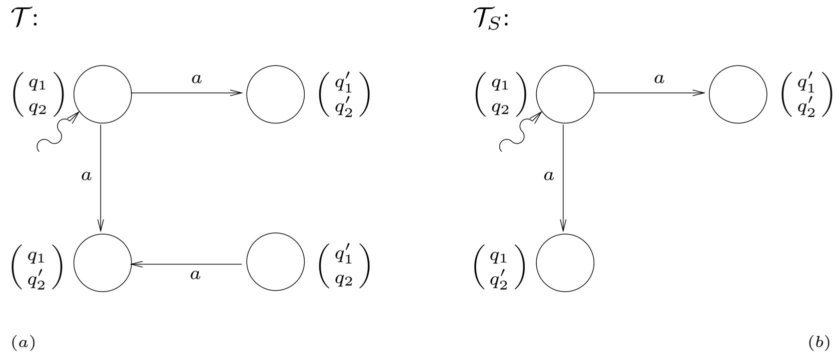


Fig. 4.19. Synchronized automaton $\mathcal{T}$ and its state-reduced version $\mathcal{T}_S$.

the resulting isolated state (q_1', q_2) we obtain the state-reduced version $\mathcal{T}_S$ of $\mathcal{T}$, which is depicted in Figure 4.19(b).

Clearly neither $\mathcal{T}$ nor $\mathcal{T}_S$ is enabling since, e.g., b is not even active in either of these synchronized automata. It is also easy to see that neither $\mathcal{T}$ nor $\mathcal{T}_S$ is deterministic since both synchronized automata contain the transition $((q_1, q_2), a, (q_1, q_2'))$ as well as the transition $((q_1, q_2), a, (q_1', q_2'))$. $\square$

Note that this example thus also suffices to conclude that the Θ-enabling (Θ-determinism) of a set of automata is not inherited by a Θ-action-reduced (Θ-transition-reduced, state-reduced) synchronized automaton over that set of automata.

Summarizing, we conclude that the automata-theoretic properties of Section 3.2 in general are not preserved from a set of automata $\mathcal{S}$ to a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$. We nevertheless show next that — under certain conditions — some of these properties *are* preserved from $\mathcal{S}$ to $\mathcal{T}$.

Reduced Versions

As before we begin by considering action reducedness, transition reducedness, and state reducedness. Note that these properties are based on the notion of reachability of states. We know from Lemma 4.6.11(2) that whenever a state q is reachable in $\mathcal{T}$, then for all $i \in \mathcal{I}$, $\mathrm{proj}_i(q)$ is reachable in $\mathcal{A}_i$. Here we study the inheritance from automata to synchronized automata. Given a state q of a synchronized automaton $\mathcal{T}$ over $\mathcal{S}$ comprising solely reachable states of the automata from $\mathcal{S}$, it is not necessarily the case that q is reachable in $\mathcal{T}$. This is because it may be the case that the transition relation of $\mathcal{T}$ allows no synchronous execution of actions from its constituting automata that would lead to q. In the following example we show that even when we consider the *maximal-free* (*maximal-ai*, *maximal-si*) synchronized automaton over $\mathcal{S}$, then this may still be the case.

Example 4.6.26. (Examples 4.6.5 and 4.6.20 continued) Note that all the states of all the automata of Examples 4.6.5 and 4.6.20, depicted in Figures 4.11 and 4.14, are reachable. Hence all these automata are state reduced.

Since in Example 4.6.5 both the *maximal-free* synchronized automaton $\mathcal{T}^{free}_{1,2}$ and the *maximal-ai* synchronized automaton $\mathcal{T}^{ai}_{2,3}$ have an empty transition relation, it is however clear that (p, q') and (q', r) are not reachable in $\mathcal{T}^{free}_{1,2}$ and $\mathcal{T}^{ai}_{2,3}$, respectively. Finally, in the *maximal-si* synchronized automaton $\mathcal{T}^{si}$ of Example 4.6.20 — depicted in Figure 4.15(b) — it is clear that (q'_1, q_2) is not reachable. Hence neither of these three maximal synchronized automata is state reduced. □

This example thus not only presents counterexamples for the preservation of reachability of states of automata from $\mathcal{S}$ to the *maximal-free* (*maximal-ai*, *maximal-si*) synchronized automaton over $\mathcal{S}$, but it also demonstrates that state reducedness of automata from $\mathcal{S}$ in general is not preserved by the *maximal-free* (*maximal-ai*, *maximal-si*) synchronized automaton over $\mathcal{S}$.

We now show that we can use the notion of loop limitedness to prove the reachability of any state q of the *maximal-free* synchronized automaton $\mathcal{T}$ over $\mathcal{S}$ that comprises solely reachable states of the automata from $\mathcal{S}$. To this aim, we extend Definition 4.6.9 by defining when $\mathcal{S}$ is *Θ-loop-limited*.

Definition 4.6.27. *$\mathcal{S}$ is Θ-*loop limited *if for all $i \in \mathcal{I}$, $\mathcal{S}$ is Θ-i-loop limited.* □

If $\mathcal{S}$ is Σ-loop limited, then we may also simply say that $\mathcal{S}$ is loop limited. Observe that whenever there exists a $k \in \mathcal{I}$ such that $\Theta \subseteq \Sigma_k \setminus (\bigcup_{i \in \mathcal{I} \setminus \{k\}} \Sigma_i)$, then $\mathcal{S}$ is Θ-loop limited. Whenever $\mathcal{S}$ is loop limited and $\mathcal{T}$ is the *maximal-free* synchronized automaton over $\mathcal{S}$, then Theorem 4.6.10 implies that all transitions of the automata from $\mathcal{S}$ are omnipresent in $\mathcal{T}$. Moreover, in all synchronizations of $\mathcal{T}$ only one automaton participates. If in addition $\mathcal{S}$ is finite, then we can thus simply reach q by executing one by one the sequences of transitions responsible for the reachability of those states constituting q.

Lemma 4.6.28. *Let $q \in Q$ be such that for all $i \in \mathcal{I}$, $proj_i(q)$ is reachable in $\mathcal{A}_i$. Then*

> *if $\mathcal{S}$ is finite and loop limited and $\delta_a = \mathcal{R}^{free}_a(\mathcal{S})$, for all $a \in \Sigma$, then q is reachable in $\mathcal{T}$.*

Proof. Let $\mathcal{S}$ be finite and loop limited and let $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$, for all $a \in \Sigma$. Now let $\#\mathcal{I} = n$, for some $n \geq 1$, and assume without loss of generality that $\mathcal{I} = [n]$. For all $i \in [n]$, we can fix a computation $\alpha_i = q_{i_0} a_{i_1} q_{i_1} a_{i_2} q_{i_2} \cdots a_{i_{m_i}} q_{i_{m_i}} \in \mathbf{C}_{\mathcal{A}_i}$ such that $m_i \geq 0$, $q_{i_0} \in I_i$, $a_{i_k} \in \Sigma_i$ and $q_{i_k} \in Q_i$, for all $k \in [m_i]$, and $q_{i_{m_i}} = \mathrm{proj}_i(q) \in Q_i$. Consequently, we define β inductively by a sequence $\beta_0, \beta_1, \ldots, \beta_n$ such that $\beta_n = \beta$ as follows.

$\beta_0 = q_0$ is defined by $\mathrm{proj}_i(q_0) = q_{i_0}$, for all $i \in [n]$. Hence $q_0 \in \prod_{i \in [n]} I_i = I$ and $\beta_0 \in \mathbf{C}_{\mathcal{T}}$. Moreover, $\pi_{\mathcal{A}_i}(\beta_0) = q_{i_0}$, for all $i \in [n]$.

$\beta_1 = \beta_0 a_{1_1} q_1 a_{1_2} q_2 \cdots a_{1_{m_1}} q_{m_1}$ is defined, for all $k \in [m_1]$, by $\mathrm{proj}_1(q_k) = q_{1_k}$ and $\mathrm{proj}_i(q_k) = \mathrm{proj}_i(q_0) = q_{i_0}$ if $1 < i \leq n$. Since $(q_{1_{k-1}}, a_{1_k}, q_{1_k}) \in \delta_1$, for all $k \in [m_1]$, $\mathcal{S}$ is loop limited, and $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$, for all $a \in \Sigma$, it follows that $\beta_1 \in \mathbf{C}_{\mathcal{T}}$, $\pi_{\mathcal{A}_1}(\beta_1) = \alpha_1 \in \mathbf{C}_{\mathcal{A}_1}$, and $\pi_{\mathcal{A}_i}(\beta_1) = q_{i_0}$, for all $1 < i \leq n$.

Now let $0 \leq \ell \leq n$ and assume that $\beta_0, \beta_1, \ldots, \beta_{\ell-1}$ are defined in such a way that $\beta_{\ell-1} \in \mathbf{C}_{\mathcal{T}}$, $\pi_{\mathcal{A}_i}(\beta_{\ell-1}) = \alpha_i \in \mathbf{C}_{\mathcal{A}_i}$, for all $i \in [\ell-1]$, and $\pi_{\mathcal{A}_i}(\beta_{\ell-1}) = q_{i_0}$, for all $\ell \leq i \leq n$.

$\beta_\ell = \beta_{\ell-1} a_{\ell_1} p_1 a_{\ell_2} p_2 \cdots a_{\ell_{m_\ell}} p_{m_\ell}$ is defined, for all $k \in [m_\ell]$, by $\mathrm{proj}_\ell(p_k) = q_{\ell_k}$, $\mathrm{proj}_i(p_k) = q_{i_{m_i}}$ if $i \in [\ell-1]$, and $\mathrm{proj}_i(p_k) = \mathrm{proj}_i(q_0) = q_{i_0}$ if $\ell < i \leq n$. Since $(q_{\ell_{k-1}}, a_{\ell_k}, q_{\ell_k}) \in \delta_\ell$, for all $k \in [m_\ell]$, $\mathcal{S}$ is loop limited, and $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$, for all $a \in \Sigma$, it follows that $\beta_\ell \in \mathbf{C}_{\mathcal{T}}$, $\pi_{\mathcal{A}_i}(\beta_\ell) = \alpha_i \in \mathbf{C}_{\mathcal{A}_i}$, for all $i \in [\ell]$, and $\pi_{\mathcal{A}_i}(\beta_\ell) = q_{i_0}$, for all $\ell < i \leq n$.

$\beta_n = \beta = q_0 b_1 q_1 b_2 q_2 \cdots b_z q_z$ is thus defined in such a way that $\beta \in \mathbf{C}_{\mathcal{T}}$ and, for all $i \in [n]$, $\pi_{\mathcal{A}_i}(\beta) = \alpha_i \in \mathbf{C}_{\mathcal{A}_i}$ and $\mathrm{proj}_i(q_z) = q_{i_{m_i}} = \mathrm{proj}_i(q)$. Hence q is reachable in $\mathcal{T}$. $\square$

An immediate consequence of Lemma 4.6.28 is that whenever $\mathcal{S}$ is a finite, loop-limited, and state-reduced set of automata, then the *maximal-free* synchronized automaton over $\mathcal{S}$ is state reduced (and thus transition reduced).

Theorem 4.6.29. *Let $\mathcal{S}$ be state reduced. Then*

> *if $\mathcal{S}$ is finite and loop limited and $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$, for all $a \in \Sigma$, then $\mathcal{T}$ is state reduced as well as transition reduced.* $\square$

It is worthwhile to notice that the requirement of every action being *maximal-free* as condition in this theorem cannot be replaced by requiring each action to be *maximal-ai* or *maximal-si* without invalidating the statement. In the following example we show this by demonstrating that the fact that $\mathcal{S}$ is state reduced (and thus transition reduced) in general does not imply that either the *maximal-ai* synchronized automaton over $\mathcal{S}$ or the *maximal-si* synchronized automaton over $\mathcal{S}$ is state reduced — nor does it imply that either of these synchronized automata is transition reduced.

Example 4.6.30. (Example 4.6.20 continued) Clearly $\mathcal{A}_1$ and $\mathcal{A}_2$ form a state-reduced (and thus transition-reduced), finite, and loop-limited set of automata. We have seen, however, that the *maximal-si* synchronized automaton $\mathcal{T}^{si}$ and the *maximal-ai* synchronized automaton $\mathcal{T}^{ai}$ both contain the transition $((q_1', q_2), a, (q_1, q_2'))$ while (q_1', q_2) is not reachable in either of these synchronized automata. Hence neither $\mathcal{T}^{si}$ nor $\mathcal{T}^{ai}$ is transition reduced (and thus neither state reduced). □

Finally, we investigate the conditions under which action reducedness is preserved from $\mathcal{S}$ to a synchronized automaton over $\mathcal{S}$. It turns out that already one action-reduced automaton $\mathcal{A}_k$ in $\mathcal{S}$ guarantees that $\mathcal{T}$ is action reduced, provided that each transition of $\mathcal{A}_k$ is omnipresent in $\mathcal{T}$.

Theorem 4.6.31. *Let $\mathcal{A}_j$ be Θ-action reduced. Then*

if each transition of $\mathcal{A}_j$ is omnipresent in $\mathcal{T}$ and $I \neq \varnothing$, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-action reduced.

Proof. Let each transition of $\mathcal{A}_j$ be omnipresent in $\mathcal{T}$ and let $I \neq \varnothing$. If $\Theta \cap \Sigma_j = \varnothing$, then there is nothing to prove. We thus assume that $a \in \Theta \cap \Sigma_j$. Then the fact that $\mathcal{A}_j$ is Θ-action reduced implies that there exists a useful transition $(p, a, p') \in \delta_j$ and a computation $p_0 a_1 p_1 a_2 p_2 \cdots a_m p_m a p' \in \mathbf{C}_{\mathcal{A}_j}$ such that $p_m = p$. Now let $q_0 \in I$ be such that $\mathrm{proj}_j(q_0) = p_0$. Then the fact that each transition of $\mathcal{A}_j$ is omnipresent in $\mathcal{T}$ implies that there exists a $(q_0, a_1, q_1) \in \delta$ such that $\mathrm{proj}_j(q_1) = p_1$. By repeating this argument we thus obtain that for all $k \in [m]$, there exists a $(q_{k-1}, a_k, q_k) \in \delta$ such that $\mathrm{proj}_j(q_{k-1}) = p_{k-1}$ and $\mathrm{proj}_j(q_k) = p_k$. This means that there exists a computation $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots a_m q_m \in \mathbf{C}_{\mathcal{T}}$ such that $\pi_{\mathcal{A}_j}(\alpha) = p_0 a_1 p_1 a_2 p_2 \cdots a_m p_m$. Since $\mathrm{proj}_j(q_m) = p_m = p$ and (p, a, p') is omnipresent in $\mathcal{T}$, there must exist a computation $\alpha a q_{m+1} \in \mathbf{C}_{\mathcal{T}}$ such that $\mathrm{proj}_j(q_{m+1}) = p'$. Hence a is active in $\mathcal{T}$ and $\mathcal{T}$ is thus $\Theta \cap \Sigma_j$-action reduced. □

Together with Theorems 4.6.3, 4.6.4, 4.6.8, and 4.6.10(2) this implies the following result.

Corollary 4.6.32. *Let $\mathcal{A}_j$ be Θ-action reduced, let $I \neq \varnothing$, and let $syn \in \{si, no\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Sigma$, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-action reduced,

(2) if $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma$, and $\mathcal{S}$ is Θ-enabling, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-action reduced, and

(3) if $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$, for all $a \in \Sigma$, and $\mathcal{S}$ is Θ-j-loop limited, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-action reduced. □

Enabling

We now turn to an investigation of the conditions under which enabling is preserved from $\mathcal{S}$ to a synchronized automaton over $\mathcal{S}$. It turns out that already one enabling automaton $\mathcal{A}_k$ in $\mathcal{S}$ guarantees that $\mathcal{T}$ is enabling, provided that each transition of $\mathcal{A}_k$ is omnipresent in $\mathcal{T}$.

Theorem 4.6.33. *Let $\mathcal{A}_j$ be Θ-enabling. Then*

if each a-transition of $\mathcal{A}_j$, for all $a \in \Theta$, is omnipresent in $\mathcal{T}$, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-enabling.

Proof. Let each a-transition of $\mathcal{A}_j$, for all $a \in \Theta$, be omnipresent in $\mathcal{T}$. If $\Theta \cap \Sigma_j = \varnothing$, then there is nothing to prove. We thus assume that $a \in \Theta \cap \Sigma_j$. Now let $q \in Q$. Since $a \in \Sigma_j$ and $\mathcal{A}_j$ is Θ-enabling we know that $a \ \mathrm{en}_{\mathcal{A}_j} \ \mathrm{proj}_j(q)$. The fact that each a-transition of $\mathcal{A}_j$, for all $a \in \Theta$, is omnipresent in $\mathcal{T}$ consequently implies that $a \ \mathrm{en}_{\mathcal{T}} \ q$. Hence $\mathcal{T}$ is $\Theta \cap \Sigma_j$-enabling. $\square$

Together with Theorems 4.6.3, 4.6.4, 4.6.8, and 4.6.10(2) this implies the following result.

Corollary 4.6.34. *Let $\mathcal{A}_j$ be Θ-enabling and let $syn \in \{si, no\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-enabling,

(2) if $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, and $\mathcal{S}$ is Θ-enabling, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-enabling, and

(3) if $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, and $\mathcal{S}$ is Θ-j-loop limited, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-enabling. $\square$

Determinism

Finally, we turn to an investigation of the conditions under which determinism is preserved from $\mathcal{S}$ to a synchronized automaton over $\mathcal{S}$. It turns out that whenever $\mathcal{S}$ is deterministic, then so is $\mathcal{T}$ provided that all its actions are *maximal-ai* or *maximal-si*.

Theorem 4.6.35. *Let $\mathcal{S}$ be Θ-deterministic and let $syn \in \{ai, si\}$. Then*

if $\delta_a \subseteq \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma$, then $\mathcal{T}$ is Θ-deterministic.

Proof. Let $a \in \Theta \cap \Sigma$ and let $\delta_a \subseteq \mathcal{R}_a^{syn}(\mathcal{S})$. Now assume there exists a $q \in Q$ such that $(q, q') \in \delta_a$ and $(q, q'') \in \delta_a$, with $q' \neq q''$. Then there must exist an $i \in \mathcal{I}$ such that $\mathrm{proj}_i(q') \neq \mathrm{proj}_i(q'')$. Now we have two possibilities.
If $\mathrm{proj}_i^{[2]}(q, q') \in \delta_{i,a}$ and $\mathrm{proj}_i^{[2]}(q, q'') \in \delta_{i,a}$, then $\mathcal{A}_i$ is not $\{a\}$-deterministic, a contradiction.
If $\mathrm{proj}_i^{[2]}(q, q') \in \delta_{i,a}$ and $\mathrm{proj}_i^{[2]}(q, q'') \notin \delta_{i,a}$ or — vice versa — $\mathrm{proj}_i^{[2]}(q, q') \notin \delta_{i,a}$ and $\mathrm{proj}_i^{[2]}(q, q'') \in \delta_{i,a}$, then $(q, q'') \notin \mathcal{R}_a^{syn}(\mathcal{S})$ or — respectively — $(q, q') \notin \mathcal{R}_a^{syn}(\mathcal{S})$, a contradiction in either way.
Hence $q' = q''$ and $\mathcal{T}$ is thus Θ-deterministic. $\square$

We note that this theorem does not cover the case of *maximal-free* synchronized automata. In fact, if $\mathcal{S}$ is Θ-deterministic, then this in general does not imply that also the *maximal-free* synchronized automaton over $\mathcal{S}$ is Θ-deterministic. This can be concluded from Example 4.6.20, where it is easy to see that $\{\mathcal{A}_1, \mathcal{A}_2\}$ is loop limited and deterministic, whereas $\mathcal{T}^{free}$ is not deterministic. This implies that neither the Θ-determinism of the $\mathcal{R}^{no}$-team automaton over $\mathcal{S}$ is implied by the Θ-determinism of $\mathcal{S}$.

4.6.3 Conclusion

This section forms a detailed, although limited, account of our initial investigation of the top-down inheritance — from synchronized automata to their (sub)automata — and the bottom-up preservation — from automata to synchronized automata — of the automata-theoretic properties from Section 3.2. The obtained results lean heavily on the presence and omnipresence of transitions of (sub)automata in synchronizations of synchronized automata over these (sub)automata. These two auxiliary notions have been treated in an intermezzo preceding our investigation.

We have focused on *maximal-free*, *maximal-ai*, and *maximal-si* synchronized automata. To a lesser degree we have moreover considered synchronized automata in which either every action is *free*, or every action is *ai*, or every action is *si*. Results on the $\mathcal{R}^{no}$-synchronized automaton over $\mathcal{S}$ have been mentioned only when they required almost no effort. Finally, the only additional conditions that have been considered in our search for sufficient conditions under which the automata-theoretic properties from Section 3.2 are inherited top-down or preserved bottom-up, are the loop limitedness and enabling of $\mathcal{S}$. Consequently, for many of these properties it remains to narrow down which combinations of specific conditions and types of (synchronized) automata guarantee their top-down inheritance and their bottom-up preservation. Furthermore, once other types of synchronization have been introduced, inheritance and preservation can be considered in the context of a broader class of synchronized automata (cf. Chapter 5).

4.7 Inheritance of Synchronizations

In the previous section we investigated the effect that the types of synchronization introduced in Sections 4.4 and 4.5 have on the inheritance of the automata-theoretic properties from Section 3.2. In this section we investigate the conditions under which these types of synchronization are themselves inherited top-down — from synchronized automata to subautomata — and preserved bottom-up — from subautomata to synchronized automata.

Note that we deal with synchronizations *between* automata constituting a synchronized automaton. There is thus no need to study whether synchronizations are inherited by automata from synchronized automata — and vice versa — since in any automaton — and in any synchronized automaton over a single automaton — all its actions trivially are *free*, *ai*, and *si*.

We begin by studying the inheritance of the types of synchronization introduced in Section 4.4. The property of an action a being *free* (ai, si) in a synchronized automaton is inherited by all its subautomata having a as one of their actions.

Lemma 4.7.1. *(1)* $\Sigma_J \cap \mathit{Free}(\mathcal{T}) \subseteq \mathit{Free}(\mathit{SUB}_J)$,

(2) $\Sigma_J \cap AI(\mathcal{T}) \subseteq AI(\mathit{SUB}_J)$, *and*

(3) $\Sigma_J \cap SI(\mathcal{T}) \subseteq SI(\mathit{SUB}_J)$.

Proof. (1) Let $a \in \Sigma_J \cap \mathit{Free}(\mathcal{T})$. Now assume that $a \notin \mathit{Free}(\mathit{SUB}_J)$. This means there must exist a transition $(p, a, p') \in \delta_J$ such that $\#\{i \in J \mid \mathrm{proj}_i^{[2]}(p, p') \in \delta_{i,a}\} > 1$. Then Theorem 4.6.2 implies that there exists a $(q, q') \in \delta_a$ such that $\mathrm{proj}_J^{[2]}(q, q') = (p, p')$, and thus $\#\{i \in \mathcal{I} \mid \mathrm{proj}_i^{[2]}(q, q') \in \delta_{i,a}\} > 1$. This contradicts the fact that a is *free* in $\mathcal{T}$. Hence $a \in \mathit{Free}(\mathit{SUB}_J)$.

(2,3) Analogous. □

Note that the proof of Lemma 4.7.1 relies heavily on the observation that in a subautomaton of a synchronized automaton no (new) transitions — i.e. other than those obtained as projections of existing transitions of the synchronized automaton — are introduced. Hence if there exists a transition in SUB_J violating the *free* (ai, si) requirement for a, then Theorem 4.6.2 implies that this transition is present in $\mathcal{T}$, i.e. there exists an "extension" of this transition in $\mathcal{T}$ which also violates the *free* (ai, si) requirement for a.

The converses of the statements of Lemma 4.7.1 in general do not hold. The reason for this resides in the fact that an action a that is not *free* in a synchronized automaton $\mathcal{T}$, *is free* in a subautomaton of $\mathcal{T}$ provided the restriction to a subset of the automata leads to dropping those automata from $\mathcal{T}$ that caused a not to be *free* in $\mathcal{T}$. The same reasoning can be applied in case a is *ai* or *si*. In the following example we demonstrate this.

Example 4.7.2. (Example 4.4.8 continued) We have seen that in synchronized automaton $\mathcal{T}^1$ action a is neither *free*, nor *ai*, nor *si*. However, in subautomaton $SUB_{\{2\}}(\mathcal{T}^1)$ — which is essentially a copy of $\mathcal{A}_2$ — action a trivially is *free*, *ai*, and *si*. □

We now demonstrate that the converses of the statements of Lemma 4.7.1 do hold if always only one automaton participates in the execution of an action, as is the case for internal actions. More general, whenever an action only belongs to automata which are included in a subautomaton, then the properties of being *free* (*ai*, *si*) are preserved from that subautomaton to the synchronized automaton as a whole.

Lemma 4.7.3. *Let* $\Sigma_J \cap (\bigcup_{i \in \mathcal{I} \setminus J} \Sigma_i) = \varnothing$. *Then*

(1) $\mathit{Free}(SUB_J) \subseteq \Sigma_J \cap \mathit{Free}(\mathcal{T})$,

(2) $AI(SUB_J) \subseteq \Sigma_J \cap AI(\mathcal{T})$, *and*

(3) $SI(SUB_J) \subseteq \Sigma_J \cap SI(\mathcal{T})$.

Proof. (1) Let $a \in \mathit{Free}(SUB_J)$. Hence $a \in \Sigma_J$. Now assume that $a \notin \mathit{Free}(\mathcal{T})$. Then there exists a transition $(q, q') \in \delta_a$ that violates the requirement for a to be *free* in $\mathcal{T}$. However, since $\Sigma_J \cap (\bigcup_{i \in \mathcal{I} \setminus J} \Sigma_i) = \varnothing$, we have that for all $i \in \mathcal{I} \setminus J$, $a \notin \Sigma_i$. We conclude that the violation of the requirement for a to be *free* in $\mathcal{T}$ thus occurs in SUB_J, i.e. $\mathrm{proj}_J^{[2]}(q, q')$ violates the requirement for a to be *free* in SUB_J, a contradiction. Hence $a \in \Sigma_J \cap \mathit{Free}(\mathcal{T})$.

(2,3) Analogous. □

Together with Lemma 4.7.1, this lemma implies the following result.

Theorem 4.7.4. *Let* $\Sigma_J \cap (\bigcup_{i \in \mathcal{I} \setminus J} \Sigma_i) = \varnothing$. *Then*

(1) $\Sigma_J \cap \mathit{Free}(\mathcal{T}) = \mathit{Free}(SUB_J)$,

(2) $\Sigma_J \cap AI(\mathcal{T}) = AI(SUB_J)$, *and*

(3) $\Sigma_J \cap SI(\mathcal{T}) = SI(SUB_J)$. □

Finally, we conclude this section with a result on the inheritance of the maximal types of synchronization introduced in Section 4.5. We show that under certain conditions, the property of an action a being *maximal-free* (*maximal-ai*, *maximal-si*) in a synchronized automaton is inherited by each subautomaton of that synchronized automaton having a as one of its actions.

Theorem 4.7.5. *Let $a \in \Sigma_J$. Then*

(1) if $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$ and $\mathcal{S}$ is $\{a\}$-loop limited, then $(\delta_J)_a = \mathcal{R}_a^{free}(\{\mathcal{A}_j \mid j \in J\})$,

(2) if $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \neq \varnothing$, then $(\delta_J)_a = \mathcal{R}_a^{ai}(\{\mathcal{A}_j \mid j \in J\})$, and

(3) if $\delta_a = \mathcal{R}_a^{si}(\mathcal{S})$, then $(\delta_J)_a = \mathcal{R}_a^{si}(\{\mathcal{A}_j \mid j \in J\})$.

Proof. (1) Let $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$ and let $\mathcal{S}$ be $\{a\}$-loop limited. Then according to Lemma 4.7.1(1) we only need to prove that $\mathcal{R}_a^{free}(\{\mathcal{A}_j \mid j \in J\}) \subseteq (\delta_J)_a$. Now let $(q, q') \in \mathcal{R}_a^{free}(\{\mathcal{A}_j \mid j \in J\})$. Then there exists a $k \in J$ such that ${\mathrm{proj}_k}^{[2]}(q, q') = (p, p') \in \delta_{k,a}$ and for all $i \in J \setminus \{k\}$, $\mathrm{proj}_i(q') = \mathrm{proj}_i(q)$. Since $\mathcal{S}$ is $\{a\}$-loop limited it follows from Theorem 4.6.10(2) that (p, a, p') is omnipresent in $\mathcal{T}$. Together with the fact that $\delta_a = \mathcal{R}_a^{free}(\mathcal{S})$ this implies that there must exist an $(r, r') \in \delta_a$ such that ${\mathrm{proj}_J}^{[2]}(r, r') = (q, q')$ and thus $(q, q') = {\mathrm{proj}_J}^{[2]}(r, r') \in (\delta_J)_a$.

(2) Let $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \neq \varnothing$. Then by Lemma 4.7.1(2) we only need to prove that $\mathcal{R}_a^{ai}(\{\mathcal{A}_j \mid j \in J\}) \subseteq (\delta_J)_a$. Now let $(q, q') \in \mathcal{R}_a^{ai}(\{\mathcal{A}_j \mid j \in J\})$. Then there exists a $K \subseteq J$ such that for all $k \in K$, $a \in \Sigma_k$, ${\mathrm{proj}_k}^{[2]}(q, q') \in \delta_{k,a}$, and for all $i \in J \setminus K$, ${\mathrm{proj}_i}^{[2]}(q, q') \notin \delta_{i,a}$ and $a \notin \Sigma_i$. Since $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \neq \varnothing$, it follows from Theorem 4.6.6 that for all $k \in K$, $(\mathrm{proj}_k(q), a, \mathrm{proj}_k(q'))$ is present in $\mathcal{T}$. Together with the fact that $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$ this implies that there must exist an $(r, r') \in \delta_a$ such that ${\mathrm{proj}_J}^{[2]}(r, r') = (q, q')$ and thus $(q, q') = {\mathrm{proj}_J}^{[2]}(r, r') \in (\delta_J)_a$.

(3) Let $\delta_a = \mathcal{R}_a^{si}(\mathcal{S})$. Then according to Lemma 4.7.1(3) we only need to prove that $\mathcal{R}_a^{si}(\{\mathcal{A}_j \mid j \in J\}) \subseteq (\delta_J)_a$. Now let $(q, q') \in \mathcal{R}_a^{si}(\{\mathcal{A}_j \mid j \in J\})$. Then there exists a $K \subseteq J$ such that for all $k \in K$, ${\mathrm{proj}_k}^{[2]}(q, q') \in \delta_{k,a}$ and for all $i \in J \setminus K$, ${\mathrm{proj}_i}^{[2]}(q, q') \notin \delta_{i,a}$ and a is not enabled in $\mathcal{A}_i$ at $\mathrm{proj}_i(q)$. From Theorem 4.6.3 it now follows that for all $k \in K$, $(\mathrm{proj}_k(q), a, \mathrm{proj}_k(q'))$ is omnipresent in $\mathcal{T}$. Together with the fact that $\delta_a = \mathcal{R}_a^{si}(\mathcal{S})$ this implies that there must exist an $(r, r') \in \delta_a$ such that ${\mathrm{proj}_J}^{[2]}(r, r') = (q, q')$ and thus $(q, q') = {\mathrm{proj}_J}^{[2]}(r, r') \in (\delta_J)_a$. □

5. Team Automata

In the preceding two chapters we have prepared the basis for team automata.

In Chapter 3 we have defined automata underlying the *component automata* that team automata are built on. In Chapter 4 we consequently defined synchronized automata over sets of automata as a way to coordinate the interactions of those automata. Team automata are defined similar to synchronized automata, but they coordinate component automata rather than automata. The extra feature of component automata with respect to automata is a classification of their set of actions into *input*, *output*, and *internal* actions. *Subteams* of team automata are defined analogous to the subautomata of synchronized automata and we show how to iteratively build team automata over team automata similar to the iterative construction of synchronized automata.

The extra feature of component automata now allows us to characterize more types of synchronization and more *predicates of synchronization* by using the classification of their sets of actions. Consequently *maximal-syn team automata* are defined with respect to a given type of synchronization *syn*, similar to the way we did this in the context of synchronized automata. Finally, also this chapter is concluded with a study of the effect that synchronizations have on the inheritance of the automata-theoretic properties introduced in Section 3.2.

5.1 Definitions

Throughout this section we will occasionally illustrate our definitions using simple examples of coffee vending machines and their customers. This class of examples is very common in the literature on formal methods. Through these examples we thus hope to facilitate an interesting comparison of the team automata framework with models such as, e.g., (Theoretical) Communicating Sequential Processes (see, e.g., [Hoa78], [BHR84], and [Hoa85]) and Input/Output automata (see, e.g., [Tut87], [LT87], [LT89], and [Lyn96]). A survey can be found in [Shi97].

5.1.1 Component Automata

Team automata are built from component automata.

A component automaton is an automaton together with a classification of its actions. The actions are divided into two main categories. *Internal* actions have strictly local visibility and can thus not be used for collaboration with other components, whereas *external* actions are observable by other components. These external actions can be used for collaboration between components and are divided into two more categories: *input* actions and *output* actions. As formulated in [Ell97]: "input actions are not under the local system's control and are caused by another non-local component, the output actions are under the system's control and are externally observable by other components, and internal actions are under the local system's control but are not externally observable".

When describing a component automaton with the system to be modeled in mind, one of the design issues that thus has to be considered is the role of the actions within that component in relation to the other components within the system.

Definition 5.1.1. *A* component automaton *is a construct* $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$*, where*

$(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ *is an automaton,*
Σ_{inp} *is the* input alphabet *of* $\mathcal{C}$*,*
Σ_{out} *is the* output alphabet *of* $\mathcal{C}$*, and*
Σ_{int} *is the* internal alphabet *of* $\mathcal{C}$ *such that* Σ_{inp}*,* Σ_{out}*, and* Σ_{int} *are mutually disjoint.* □

The automaton $(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ of a component automaton $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ is called the *underlying automaton* of $\mathcal{C}$ and it is denoted by und($\mathcal{C}$). Moreover, the elements of the input, output, and internal alphabet of $\mathcal{C}$ are called the *input*, *output*, and *internal actions* of $\mathcal{C}$, respectively. We refer to $\mathcal{C}$ as the *trivial component automaton* if $\mathcal{C} = (\varnothing, (\varnothing, \varnothing, \varnothing), \varnothing, \varnothing)$. Finally, if both Q and $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$ are finite, then $\mathcal{C}$ is called a *finite component automaton*.

Definition 5.1.2. *Let* $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *be a component automaton. Then*

(1) the (full) alphabet *of* $\mathcal{C}$ *is denoted by* Σ *and is defined as* $\Sigma = \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$*,*

(2) the external alphabet *of* $\mathcal{C}$ *is denoted by* Σ_{ext} *and is defined as* $\Sigma_{ext} = \Sigma_{inp} \cup \Sigma_{out}$*, and*

(3) the locally-controlled alphabet *of* $\mathcal{C}$ *is denoted by* Σ_{loc} *and is defined as* $\Sigma_{loc} = \Sigma_{out} \cup \Sigma_{int}$. □

The elements of the full alphabet of a component automaton $\mathcal{C}$ are called the *actions* of $\mathcal{C}$. The elements of the external and locally-controlled alphabets are called the *external* and *locally-controlled actions* of $\mathcal{C}$, respectively.

For a given component automaton $\mathcal{C}$, its set of (*finite* and *infinite*) *computations* and — given a set of actions Θ — its Θ-*records* and its Θ-*behavior* are carried over from Definitions 3.1.2 and 3.1.7 through its underlying automaton und($\mathcal{C}$). This means that we have, e.g., $\mathbf{C}_{\mathcal{C}} = \mathbf{C}_{\mathrm{und}(\mathcal{C})}$ and $\mathbf{B}^{\Theta}_{\mathcal{C}} = \mathbf{B}^{\Theta}_{\mathrm{und}(\mathcal{C})}$.

The different roles actions can play within a component automaton naturally give rise to various behavioral language definitions. Given a component automaton $\mathcal{C}$, we can distinguish specific *records* and *behavior* of $\mathcal{C}$ by selecting an appropriate subset of Σ.

If $\Theta = \Sigma_{inp}$, then we refer to the Θ-records of $\mathcal{C}$ as the *input records* and to $\mathbf{B}^{\Theta,\infty}_{\mathcal{C}}$ as the *input behavior* of $\mathcal{C}$. Analogously, by setting $\Theta = \Sigma_{out}$, we obtain the *output records* and the *output behavior* of $\mathcal{C}$; with $\Theta = \Sigma_{int}$ we deal with *internal records* and the *internal behavior* of $\mathcal{C}$; in case $\Theta = \Sigma_{ext}$ we have *external records* and the *external behavior* of $\mathcal{C}$; finally, when $\Theta = \Sigma_{loc}$ we have *locally-controlled records* and the *locally-controlled behavior* of $\mathcal{C}$. Needless to say, also *finitary* and *infinitary* (Θ-)*behavior* can be distinguished in this way.

Example 5.1.3. Let $\mathcal{C} = (\{e,f\}, (\{\$\}, \{c\}, \varnothing), \{(e,\$,f),(f,c,e)\}, \{e\})$ be a component automaton modeling a very simple **c**offee vending machine. It is depicted in Figure 5.1.

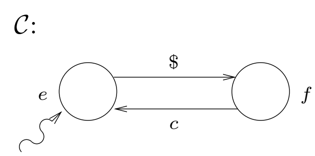


Fig. 5.1. Component automaton $\mathcal{C}$.

State e indicates that the coin slot of the vending machine is **e**mpty, while state f indicates that it is **f**illed. The result of inserting a dollar is modeled by the action \$ and fills the coin slot. The vending machine obviously is not in charge of determining the moment a dollar is inserted and \$ is thus defined to be an input action. The automaton does decide when to output coffee and this should moreover be observable by the environment. Hence the result of outputting a **c**offee is modeled by the output action c. After the vending machine has produced the coffee it is ready for another request for coffee. Initially, the vending machine is waiting for the insertion of a dollar into its empty coin slot. Hence the vending machine's initial state is e.

The behavior of the vending machine is alternatingly accepting a dollar and producing a coffee. It can do so ad infinitum. □

Before we turn to the definition of a team automaton formed from a set of component automata we fix some notation.

Notation 4. *In the rest of this chapter we assume a fixed, but arbitrary and possibly infinite index set* $\mathcal{I} \subseteq \mathbb{N}$*, which we will now use to index the component automata involved. For each* $i \in \mathcal{I}$*, we let* $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$ *be a fixed component automaton and we use* Σ_i *to denote its set of actions* $\Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$*. Moreover, we let* $\mathcal{S} = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$ *be a fixed set of component automata. Recall that* $\mathcal{I} \subseteq \mathbb{N}$ *implies that* $\mathcal{I}$ *is ordered by the usual* $\leq$ *relation on* $\mathbb{N}$*, thus inducing an ordering on* $\mathcal{S}$*. Note that the* $\mathcal{C}_i$ *are not necessarily different.* □

5.1.2 Team Automata

When composing a team automaton over $\mathcal{S}$, we require that the internal actions of the component automata involved are private, i.e. uniquely associated to one component automaton. This is formally expressed as follows.

Definition 5.1.4. $\mathcal{S}$ *is a* composable system *if for all* $i \in \mathcal{I}$,

$$\Sigma_{i,int} \cap \bigcup_{j \in \mathcal{I} \setminus \{i\}} \Sigma_j = \varnothing .$$ □

Note that every subset of a composable system is again a composable system.

Example 5.1.5. (Example 5.1.3 continued) Let $\mathcal{A} = (\{s,t\}, (\{c\}, \{\$\}, \varnothing), \{(s,\$,t), (t,c,s)\}, \{s\})$ be a component automaton modeling a coffee **a**ddict. It is depicted in Figure 5.2.

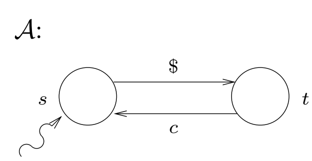


Fig. 5.2. Component automaton $\mathcal{A}$.

State s indicates that our coffee addict is (temporarily) **s**atisfied, while state t indicates that our coffee addict is **t**hirsty (again). The result of our coffee addict inserting a dollar (into a coffee vending machine) is modeled by the action \$ and shows our coffee addict's thirst. Our coffee addict obviously is in charge of determining when to show his or her thirst and thus when to insert a dollar. Since this should also be observable by the coffee vending machine we define \$ to be an output action. Our coffee addict however cannot decide when the coffee vending machine produces the much-awaited coffee. The result of our coffee addict trenching his or her thirst and becoming satisfied is thus modeled by the input action c. Initially our coffee addict is satisfied, modeled by our coffee addict's initial state s.

The behavior of our coffee addict is alternatingly inserting a dollar and trenching his or her thirst with a delicious cup of coffee. Like a true addict, our coffee addict can do so ad infinitum.

Since neither $\mathcal{C}$ nor $\mathcal{A}$ has any internal actions, $\mathcal{C}$ and $\mathcal{A}$ trivially form a composable system $\{\mathcal{C}, \mathcal{A}\}$. □

We are now ready to define a team automaton over a composable system $\mathcal{S}$ as a synchronized automaton over $\mathcal{S}$, except that in our definition of a team automaton we need to specify how to deal with the distinction of the alphabet into input, output, and internal actions.

The alphabet of actions of any team automaton $\mathcal{T}$ formed from $\mathcal{S}$ is uniquely determined by the alphabets of actions of the component automata constituting $\mathcal{S}$. The internal actions of the component automata will be the internal actions of $\mathcal{T}$. Each action which is output for one or more of the component automata is an output action of $\mathcal{T}$. Hence an action that is an output action of one component automaton and also an input action of another component automaton, is considered an output action of the team automaton. The input actions of the component automata that do not occur at all as an output action of any of the component automata, are the input actions of the team automaton. The reason for this construction of alphabets is again based on the intuitive idea of [Ell97] that when relating an input action a of a component automaton to an output action a of another component, then the input may be thought of as being caused by the output. On the other hand, output actions remain observable as output to other component automata.

Finally, the freedom of choosing a particular transition relation for a synchronized automaton over $\mathcal{S}$ is reduced slightly in the definition of a team automaton over $\mathcal{S}$, viz. for an internal action each component automaton always retains all its possibilities to execute that action and change state. Since $\mathcal{S}$ is a composable system, all internal actions are moreover uniquely associated to one component automaton, which implies that synchronizations on internal actions thus never involve more than one component automaton.

Definition 5.1.6. *Let $\mathcal{S}$ be a composable system. Then a* team automaton *over $\mathcal{S}$ is a construct $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$, where*

$(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ is a synchronized automaton over $\mathcal{S}$ such that

$$\delta_a = \Delta_a(\mathcal{S}), \text{ for all } a \in \Sigma_{int},$$

$\Sigma_{inp} = (\bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}) \setminus \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$,
$\Sigma_{out} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$, *and*
$\Sigma_{int} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,int}$. □

The synchronized automaton $(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ of a team automaton $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ is called the *underlying synchronized automaton* of $\mathcal{T}$ and it is denoted by und$(\mathcal{T})$.

All team automata over a given composable system have the same set of states, the same alphabet of actions — including the distribution over input, output, and internal actions — and the same set of initial states. They only differ by the choice of the transition relation, and in fact only as far as external actions are concerned: for each external action a we have the freedom to choose a δ_a. This implies that $\mathcal{S}$, even if it is a composable system, does not uniquely define a team automaton.

Example 5.1.7. (Example 5.1.5 continued) We now show how our coffee addict can obtain a coffee from our vending machine by forming a team automaton $\mathcal{T}$ over the composable system $\{\mathcal{C}, \mathcal{A}\}$. This team automaton should model a form of collaboration between our coffee addict and the vending machine. This is implemented by synchronizations of certain actions. We require the output action \$ of our coffee addict to be synchronized with the input action \$ of our vending machine. The occurrence of this action in the team automaton then reflects the simultaneous execution of \$ by our coffee addict and our vending machine. Likewise action c is simultaneously executed by our coffee addict and our vending machine. This defines the transition relation of $\mathcal{T}$. Note that only the transition relation of $\mathcal{T}$ had to be chosen, the other elements of $\mathcal{T}$ follow directly from Definition 5.1.6. Note in particular that both \$ and c are output actions of $\mathcal{T}$. Hence $\mathcal{T}$ is formally defined as $\mathcal{T} = (\{(e,s), (e,t), (f,s), (f,t)\}, (\varnothing, \{\$, c\}, \varnothing), \delta, \{(e,s)\})$, where $\delta = \{((e,s), \$, (f,t)), ((f,t), c, (e,s))\}$. It is depicted in Figure 5.3. □

Consistency in the sense that in a team automaton every action appears exclusively as an input, output, or internal action, is guaranteed by Definition 5.1.6 (which ensures that input and output actions remain distinct) and

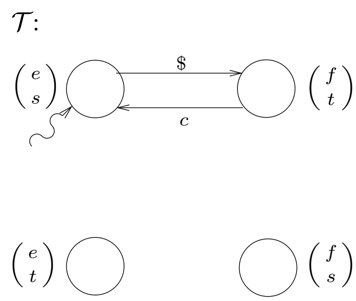


Fig. 5.3. Team automaton $\mathcal{T}$ over $\{\mathcal{C}, \mathcal{A}\}$.

the fact that a team automaton is constructed over a composable system. Together with Definition 5.1.4 this implies that every team automaton is again a component automaton, which in its turn could be used as a component automaton in a new team automaton.

Theorem 5.1.8. *Every team automaton is a component automaton.* □

As was the case for synchronized automata (cf. Section 4.1) we note that even though a team automaton over a composable system consisting of just one component automaton $\{\mathcal{C}_i\}$ is again a component automaton, such a team automaton is different from its only constituting component automaton.

All observations on (component) automata hold for team automata as well. The abbreviations for sets of alphabets carry over to team automata in the obvious way. Finally, note that whenever the distinction of the alphabet of actions into input, output, and internal actions is irrelevant, then a synchronized automaton can be seen as a team automaton. As a matter of fact, in examples in the remainder of this chapter we will often refer to synchronized automata defined in earlier chapters as team automata.

5.1.3 Subteams

Similar to the way we extracted subautomata from synchronized automata, by focusing on a subset of the composable system $\mathcal{S}$ of component automata constituting a team automaton $\mathcal{T}$ we now distinguish *subteams* within $\mathcal{T}$. As before, the transitions of a subteam are restrictions of the transitions of $\mathcal{T}$ to the component automata in the subteam, while its actions are the actions of the component automata involved. However, the actions of both component automata and team automata are distributed over three distinct alphabets. Since we want to be able to deal with a subteam as an independent team automaton over a subset of $\mathcal{S}$, we need to classify its actions without the context provided by $\mathcal{T}$. Hence, whether an action is input, output, or internal for the subteam only depends on its role in the component automata forming the subteam rather than on how it is classified in $\mathcal{T}$. This means in particular that an action which is an output action of $\mathcal{T}$ is an input action for the subteam, whenever this action is an input action of at least one of the component automata of the subteam and no component automata of the subteam have this action as an output action.

Definition 5.1.9. *Let* $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *be a team automaton over the composable system* $\mathcal{S}$ *and let* $J \subseteq \mathcal{I}$. *Then the* subteam *of* $\mathcal{T}$ determined by J *is denoted by* $SUB_J(\mathcal{T})$ *and is defined as* $SUB_J(\mathcal{T}) = (Q_J, (\Sigma_{J,inp}, \Sigma_{J,out}, \Sigma_{J,int}), \delta_J, I_J)$, *where*

$(Q_J, \Sigma_{J,inp} \cup \Sigma_{J,out} \cup \Sigma_{J,int}, \delta_J, I_J)$ *is the subautomaton* $SUB_J(und(\mathcal{T}))$,
$\Sigma_{J,inp} = (\bigcup_{j\in J} \Sigma_{j,inp}) \setminus \bigcup_{j\in J} \Sigma_{j,out}$,
$\Sigma_{J,out} = \bigcup_{j\in J} \Sigma_{j,out}$, *and*
$\Sigma_{J,int} = \bigcup_{j\in J} \Sigma_{j,int}$. □

As before, we write SUB_J instead of $SUB_J(\mathcal{T})$ whenever $\mathcal{T}$ is clear from the context. Note that the notation SUB_J is used both for the subautomaton of a synchronized automaton and for the subteam of a team automaton. In cases where this might lead to confusion, we will always state explicitly the type of automaton we deal with.

It is not hard to see that any subteam satisfies the requirements of a team automaton.

Theorem 5.1.10. *Let* $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *be a team automaton over the composable system* $\mathcal{S}$ *and let* $J \subseteq \mathcal{I}$. *Then*

SUB_J *is a team automaton over* $\{\mathcal{C}_j \mid j \in J\}$.

Proof. We already noted that every subset of a composable system is again a composable system. Since the alphabets of SUB_J as given in Definition 5.1.9 moreover satisfy the requirements of Definition 5.1.6 for team automata over $\{\mathcal{C}_j \mid j \in J\}$, it directly follows from Theorem 4.1.8 that SUB_J is a team automaton over $\{\mathcal{C}_j \mid j \in J\}$. □

Similar to our conclusion — in Subsection 4.1.2 — that a subautomaton of a synchronized automaton is again a synchronized automaton, and thus also an automaton, we now conclude from Theorem 5.1.10 that a subteam of a team automaton is again a team automaton and thus, by Theorem 5.1.8, also a component automaton. Based on the results from Section 4.3 we will consider the dual approach and use team automata as component automata in "larger" team automata in the next section.

5.2 Iterated Composition

This section continues our investigation of Section 4.3, the difference being that instead of synchronized automata we now consider team automata. This means that we have to take into account that team automata can only be formed over composable systems and, moreover, that we deal with three mutually disjoint alphabets constituting the alphabet of a team automaton.

Notation 5. *In the rest of this chapter we let $\mathcal{S}$ be a composable system.* □

We consider the issue of iteratively composing team automata, given a composable system of team automata. First we prove that composability is preserved in the process of iteration.

Theorem 5.2.1. *Let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, form a partition of $\mathcal{I}$. Let, for each $j \in \mathcal{J}$, $\mathcal{T}_j$ be a team automaton over $\mathcal{S}_j = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. Then*

$\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is a composable system.

Proof. Denote for each $\mathcal{T}_j$, $j \in \mathcal{J}$, by Γ_j its set of actions and by $\Gamma_{j,int}$ its internal alphabet. By Definition 5.1.6 we have $\Gamma_{j,int} = \bigcup_{i \in \mathcal{I}_j} \Sigma_{i,int}$ and $\Gamma_j = \bigcup_{i \in \mathcal{I}_j} \Sigma_i$, for all $j \in \mathcal{J}$. By the composability of $\mathcal{S}$ we have $\Sigma_{i,int} \cap \bigcup_{\ell \in \mathcal{I} \setminus \{i\}} \Sigma_\ell = \varnothing$, for all $i \in \mathcal{I}$. Since the $\mathcal{I}_j$ are mutually disjoint it now follows immediately that for all $j \in \mathcal{J}$, $\Gamma_{j,int} \cap \bigcup_{\ell \in \mathcal{J} \setminus \{j\}} \Gamma_\ell = \varnothing$. Hence $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is a composable system. □

Given a composable system one may thus form team automata over disjoint subsets of the composable system. These team automata together with the component automata not involved in any of these team automata form — by Theorem 5.2.1 — again a composable system, which can subsequently be used as the basis for the formation of still higher-level team automata. Completely analogous to Definition 4.3.8 we now define iterated team automata as a generalization of team automata.

Definition 5.2.2. $\mathcal{T}$ *is an* iterated team automaton over $\mathcal{S}$ *if either*

(1) $\mathcal{T}$ *is a team automaton over* $\mathcal{S}$*, or*

(2) $\mathcal{T}$ *is a team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where each $\mathcal{T}_j$ is an iterated team automaton over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$, for some $\mathcal{I}_j \subseteq \mathcal{I}$, and $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$.* □

As was the case for iterated synchronized automata, we see that an iterated team automaton is thus a generalization of a team automaton: every team automaton over a given composable system may also be viewed as an iterated team automaton over that composable system. Conversely, as before, team automata formed iteratively over a composable system are essentially team automata over that composable system. Once again, the only difference is the ordering and grouping of the elements from the composable system. Heavily based on the results from Section 4.3, we now formalize this statement.

By Lemma 4.3.9, the set of (initial) states of an iterated team automaton over $\mathcal{S}$ is — after reordering — the same as the set of (initial) states of any team automaton over $\mathcal{S}$. According to Lemma 4.3.10 also its actions are the same as the actions of any team automaton formed over $\mathcal{S}$. However, the basic difference between team automata and synchronized automata is the distinction of actions into three mutually disjoint alphabets. The following lemma shows that this property is not destroyed by iteration.

Lemma 5.2.3. *Let $\mathcal{T} = (P, (\Gamma_{inp}, \Gamma_{out}, \Gamma_{int}), \gamma, J)$ be an iterated team automaton over $\mathcal{S}$. Then*

(1) $\Gamma_{inp} = (\bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}) \setminus \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$,

(2) $\Gamma_{out} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$, *and*

(3) $\Gamma_{int} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,int}$.

Proof. If $\mathcal{T}$ is a team automaton over $\mathcal{S}$, then the statement follows immediately from Definition 5.1.6. Now assume that $\mathcal{T}$ is a team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, and each $\mathcal{T}_j = (P_j, (\Gamma_{j,inp}, \Gamma_{j,out}, \Gamma_{j,int}), \gamma_j, J_j)$ is an iterated team automaton over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$, with $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forming a partition of $\mathcal{I}$. Assume furthermore inductively that for all $j \in \mathcal{J}$, $\Gamma_{j,inp} = (\bigcup_{i\in\mathcal{I}_j} \Sigma_{i,inp}) \setminus \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,out}$, $\Gamma_{j,out} = \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,out}$, and $\Gamma_{j,int} = \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,int}$.

Then $\Gamma_{int} = \bigcup_{j\in\mathcal{J}} \Gamma_{j,int} = \bigcup_{j\in\mathcal{J}} \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,int} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,int}$, by Definition 5.1.6, and because $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$.
Similarly, $\Gamma_{out} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$.
Finally, $\Gamma_{inp} = (\bigcup_{j\in\mathcal{J}} \Gamma_{j,inp}) \setminus \Gamma_{out}$ by Definition 5.1.6. Hence $\Gamma_{inp} = (\bigcup_{j\in\mathcal{J}}((\bigcup_{i\in\mathcal{I}_j} \Sigma_{i,inp}) \setminus \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,out})) \setminus \Gamma_{out} = (\bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}) \setminus \Gamma_{out}$ because $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$. □

Hence the set of actions — including their distribution over input, output, and internal actions — of every iterated team automaton over $\mathcal{S}$ is the same as that of any team automaton over $\mathcal{S}$. Finally, from Lemma 4.3.10 we moreover know that the transitions of any team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ are — after reordering — the transitions of a team automaton over $\mathcal{S}$. Iteration in the construction of a team automaton thus does not lead to an increase of the possibilities for synchronization. In other words, we can conclude that every iterated team automaton over a composable system can be interpreted as a team automaton over that composable system by reordering its state space and its transition space.

Definition 5.2.4. *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be an iterated team automaton over $\mathcal{S}$. Then the* reordered version of $\mathcal{T}$ w.r.t. $\mathcal{S}$ *is denoted by* $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$ *and is defined as*

$$\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}} = (\{\langle q\rangle_Q \mid q \in Q\}, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \{(\langle q\rangle_Q, a, \langle q'\rangle_Q) \mid q, q' \in Q,\ (q, a, q') \in \delta\}, \{\langle q\rangle_I \mid q \in I\}). \ \square$$

Note that the notation $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$ is used both for the reordered version of a synchronized automaton and for the reordered version of a team automaton. In cases where this might lead to confusion, we will always state explicitly the type of automaton we deal with.

From Lemmata 4.3.9, 4.3.10, and 5.2.3 we conclude that $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$ indeed is a team automaton over $\mathcal{S}$ whenever $\mathcal{T}$ is an iterated team automaton over $\mathcal{S}$. In fact, $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$ is the interpretation of $\mathcal{T}$ as a team automaton over $\mathcal{S}$ by reordering. We thus obtain the following direct consequences of Theorems 4.3.12 and 4.3.13.

Theorem 5.2.5. *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be an iterated team automaton over $\mathcal{S}$ and let Θ be an alphabet disjoint from Q. Then*

(1) $\mathbf{C}^{\infty}_{\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}} = \{\langle q_0\rangle_Q a_1 \langle q_1\rangle_Q a_2 \langle q_2\rangle_Q \cdots \mid q_0 a_1 q_1 a_2 q_2 \cdots \in \mathbf{C}^{\infty}_{\mathcal{T}}\}$ *and*

(2) $\mathbf{B}^{\Theta,\infty}_{\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}} = \mathbf{B}^{\Theta,\infty}_{\mathcal{T}}$. $\square$

Theorem 5.2.6. *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a team automaton over $\mathcal{S}$ and let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, form a partition of $\mathcal{I}$. Let, for each $j \in \mathcal{J}$, $\mathcal{T}_j = (P_j, (\Gamma_{j,inp}, \Gamma_{j,out}, \Gamma_{j,int}), \gamma_j, J_j)$ be an iterated team over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. Then*

(1) if $(\delta_{\mathcal{I}_j})_a \subseteq \{(\langle q\rangle_{P_j}, \langle q'\rangle_{P_j}) \mid (q, q') \in \gamma_{j,a}\}$, for all $a \in \Gamma_{j,inp} \cup \Gamma_{j,out} \cup \Gamma_{j,int}$ for all $j \in \mathcal{J}$, then there exists a team automaton $\widehat{\mathcal{T}}$ over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ such that $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_{\mathcal{S}} = \mathcal{T}$, and

(2) if $\widehat{\mathcal{T}}$ is a team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, then $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_{\mathcal{S}} = \mathcal{T}$ implies that $(\delta_{\mathcal{I}_j})_a \setminus \{(p, p) \mid (p, p) \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_j\})\} \subseteq \{(\langle q\rangle_{P_j}, \langle q'\rangle_{P_j}) \mid (q, q') \in \gamma_{j,a}\}$, for all $a \in \Gamma_{j,inp} \cup \Gamma_{j,out} \cup \Gamma_{j,int}$ for all $j \in \mathcal{J}$. $\square$

Similar to the conclusion we reached for synchronized automata in Section 4.3 we now see that not only every iterated team automaton over $\mathcal{S}$ can be considered as a team automaton directly constructed from $\mathcal{S}$ by Definition 5.2.4, but according to Theorem 5.2.6 also every team automaton can be iteratively constructed from its subteams. Consequently, both subteams and iterated team automata can be treated as team automata — including the considerations concerning their computations and their behavior — and it thus suffices to study only the relationship between subteams and team automata in the sequel, i.e. without considering iterated team automata explicitly.

5.3 Synchronizations

In Section 4.4 we introduced three natural types of synchronization. These types of synchronization can be studied in the context of team automata as well. However, they obviously ignore whether actions are input, output, or internal to certain component automata. For internal actions which belong to only one component automaton, distinguishing between their roles in different component automata is indeed not very relevant. External actions, however, may be input to some component automata, and output to other component automata. In this section we thus investigate types of synchronizations relating to the different roles that an action may have in different component automata.

Notation 6. *For the remainder of this chapter we let* $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *be a fixed team automaton over* $\mathcal{S}$*. Note that* Σ_{inp}*,* Σ_{out}*, and* Σ_{int} *are the input, output, and internal alphabet, respectively, of any team automaton over* $\mathcal{S}$ *(i.e. not only of* $\mathcal{T}$*). Furthermore, we use* Σ *to denote the set of actions* $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$*, we use* Σ_{ext} *to denote the set of external actions* $\Sigma_{inp} \cup \Sigma_{out}$*, and we use* Σ_{loc} *to denote the set of locally-controlled actions* $\Sigma_{out} \cup \Sigma_{int}$ *of any team automaton over* $\mathcal{S}$ *(i.e. including* $\mathcal{T}$*).* □

First we separate the output role of external actions from their input role. Given an external action, we locate its input and output domain within $\mathcal{I}$, and then use these domains to define input subteams and output subteams. Finally, we define two specific types of synchronization relating such input subteams and output subteams of team automata.

Definition 5.3.1. *Let* $a \in \Sigma_{ext}$*. Then*

(1) $\mathcal{I}_{a,inp}(\mathcal{S}) = \{j \in \mathcal{I} \mid a \in \Sigma_{j,inp}\}$ *is the* input domain *of* a *in* $\mathcal{S}$ *and*

(2) $\mathcal{I}_{a,out}(\mathcal{S}) = \{j \in \mathcal{I} \mid a \in \Sigma_{j,out}\}$ *is the* output domain *of* a *in* $\mathcal{S}$*.* □

No external action of any team automaton $\mathcal{T}$ will ever be both an input and an output action for one component automaton. Thus, for each $j \in \mathcal{I}$, $\Sigma_{j,inp} \cap \Sigma_{j,out} = \varnothing$, and consequently $\mathcal{I}_{a,inp}(\mathcal{S}) \cap \mathcal{I}_{a,out}(\mathcal{S}) = \varnothing$, for all $a \in \Sigma_{ext}$.

Note that, by Definition 5.1.6, $a \in \Sigma_{out}$ if and only if $\mathcal{I}_{a,out}(\mathcal{S}) \neq \varnothing$, while $a \in \Sigma_{inp}$ if and only if $\mathcal{I}_{a,inp}(\mathcal{S}) \neq \varnothing$ and $\mathcal{I}_{a,out}(\mathcal{S}) = \varnothing$.

In the following example we show how to to determine the input and output domains of actions in a composable system.

Example 5.3.2. (Example 4.1.5 continued) We turn the automata W_i, with $i \in [4]$, into component automata by distributing their alphabet $\{a, b\}$ over input, output, and internal alphabets. We let a and b be output actions in both W_1 and W_2 and we let them be input actions in both W_3 and W_4. Since $\{W_1, W_2\}$ is now a composable system, the synchronized automaton $\mathcal{T}_{\{1,2\}}$ (over $\{W_1, W_2\}$) is now a team automaton. Likewise $\{\mathcal{T}_{\{1,2\}}, W_3, W_4\}$ is now a composable system and the synchronized automaton $\mathcal{T}$ (over $\{\mathcal{T}_{\{1,2\}}, W_3, W_4\}$) is now a team automaton. Both these team automata have an empty input alphabet, output alphabet $\{a, b\}$, and an empty internal alphabet.

Let $\mathcal{T}_1 = \mathcal{T}_{\{1,2\}}$, $\mathcal{T}_2 = W_3$, and $\mathcal{T}_3 = W_4$. Then $\mathcal{T}$ is a team automaton over $\mathcal{S} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$. Actions a and b are output actions in $\mathcal{T}_1$, whereas they are input actions in both $\mathcal{T}_2$ and $\mathcal{T}_3$. Hence $\mathcal{I}_{a,out}(\mathcal{S}) = \{1\}$ and $\mathcal{I}_{a,inp}(\mathcal{S}) = \{2, 3\}$. □

Note that the input domain and the output domain of an external action of a team automaton may be empty. For every external action, however, at least one of these domains is nonempty. In case the input (output) domain is empty, then the input (output) subteam is the trivial component automaton.

Example 5.3.3. In Figure 5.4 the structure of a team automaton $\mathcal{T}$ with respect to one of its external actions a is depicted. Indicated are its input subteam $SUB_{a,inp}$ and its output subteam $SUB_{a,out}$. The square boxes in this figure denote component automata. Clearly, $\mathcal{T}$ may also contain component automata that do not have a as an external action. □

Notation 7. *For the remainder of this chapter we make no more explicit references to the fixed composable system $\mathcal{S}$ when denoting the input and output domain of an action a in $\mathcal{S}$, i.e. we write $\mathcal{I}_{a,inp}$ and $\mathcal{I}_{a,out}$ rather than $\mathcal{I}_{a,inp}(\mathcal{S})$ and $\mathcal{I}_{a,out}(\mathcal{S})$, respectively. Furthermore, for all $a \in \Sigma_{ext}$, we use $SUB_{a,inp}(\mathcal{T})$ to denote $SUB_{\mathcal{I}_{a,inp}}(\mathcal{T})$, the* input subteam *of a in $\mathcal{T}$, and we use $SUB_{a,out}(\mathcal{T})$ to denote $SUB_{\mathcal{I}_{a,out}}(\mathcal{T})$, the* output subteam *of a in $\mathcal{T}$. If no confusion arises we even omit the $\mathcal{T}$ and simply write $SUB_{a,inp}$ and $SUB_{a,out}$, respectively.* □

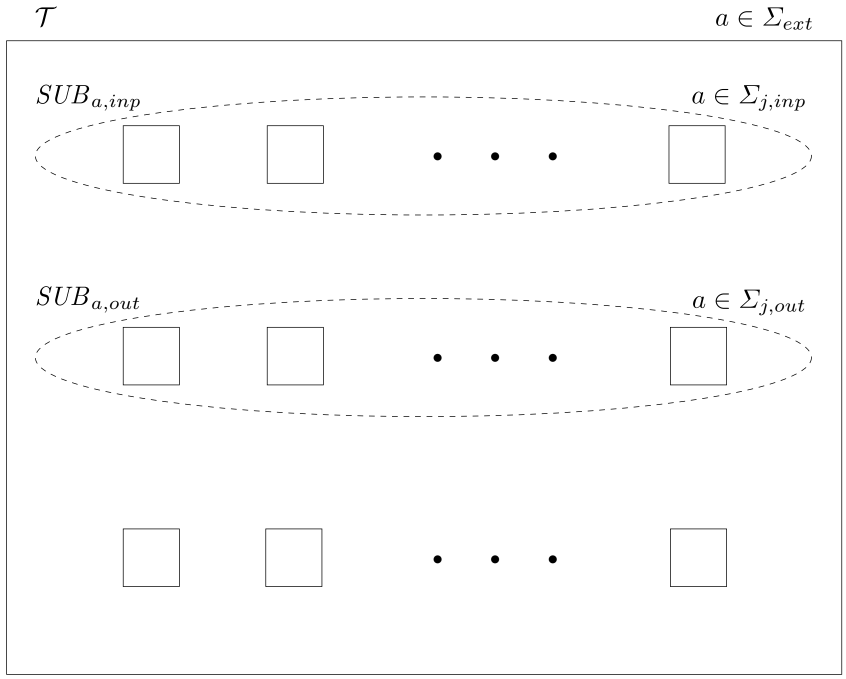


Fig. 5.4. A team automaton $\mathcal{T}$ with its subteams $SUB_{a,inp}$ and $SUB_{a,out}$.

5.3.1 Peer-to-Peer

Having determined for each external action a its input and its output subteam, we can now identify certain types of synchronization relating to a in its role as input or output. We begin by looking within these subteams, in which a by definition has only one role and all component automata are peers, in the sense that they are on an equal footing with respect to a. We say that an input (output) action a is *input (output) peer-to-peer* if every execution of a involving component automata of that subteam requires the participation of all.

This obligation to participate can be explained in a strong and in a weak sense. *Strong input (output) peer-to-peer* simply means that no synchronizations on a can take place unless all component automata in the input (output) domain of a take part. *Weak input (output) peer-to-peer* means that synchronizations on a involve all of the component automata in the input (output) domain of a which are ready to execute a — i.e. which are in a state in which a is enabled. Thus the notion of strong input (output) peer-to-peer requires that a is *ai* in its input (output) subteam, while the notion of weak input (output) peer-to-peer requires that a is *si* in its input (output) subteam.

Definition 5.3.4. *(1) The set of* strong input peer-to-peer *(sipp for short) actions of $\mathcal{T}$ is denoted by $SIPP(\mathcal{T})$ and is defined as*

$$SIPP(\mathcal{T}) = \{a \in \Sigma_{ext} \mid a \in AI(SUB_{a,inp})\},$$

(2) the set of weak input peer-to-peer *(wipp for short) actions of $\mathcal{T}$ is denoted by $WIPP(\mathcal{T})$ and is defined as*

$$WIPP(\mathcal{T}) = \{a \in \Sigma_{ext} \mid a \in SI(SUB_{a,inp})\},$$

(3) the set of strong output peer-to-peer *(sopp for short) actions of $\mathcal{T}$ is denoted by $SOPP(\mathcal{T})$ and is defined as*

$$SOPP(\mathcal{T}) = \{a \in \Sigma_{ext} \mid a \in AI(SUB_{a,out})\}, \text{ and}$$

(4) the set of weak output peer-to-peer *(wopp for short) actions of $\mathcal{T}$ is denoted by $WOPP(\mathcal{T})$ and is defined as*

$$WOPP(\mathcal{T}) = \{a \in \Sigma_{ext} \mid a \in SI(SUB_{a,out})\}. \qquad \square$$

We should remark here that an external action a that does not occur as an input action in any of the component automata (implying that $\mathcal{I}_{a,inp} = \varnothing$ and that $SUB_{a,inp}$ is the trivial component automaton) can neither be *sipp* nor *wipp*. This is due to the fact that trivial component automata (as was the case for trivial automata) have no actions whatsoever, and thus neither *ai* nor *si* actions. Note that $a \in SIPP(\mathcal{T})$ or $a \in WIPP(\mathcal{T})$ does not imply that $a \in \Sigma_{inp}$. Similarly, if a is *sopp* or *wopp* in $\mathcal{T}$, then it must be the case that it occurs as an output action in at least one component automaton of $\mathcal{T}$ (implying that $a \in \Sigma_{out}$).

Note that an external action of a team automaton $\mathcal{T}$ over $\mathcal{S}$ can be both *sipp* and *sopp* in $\mathcal{T}$. In that case the external action is an input action of one component automaton of $\mathcal{S}$ and an output action of another component automaton of $\mathcal{S}$.

Example 5.3.5. (Example 5.3.3 continued) As depicted in Figures 5.5 and 5.6, strong and weak input (output) peer-to-peer synchronizations relate to synchronizations within the corresponding input (output) subteam. $\square$

Next we present a more concrete example of strong and weak input (output) synchronizations within team automata.

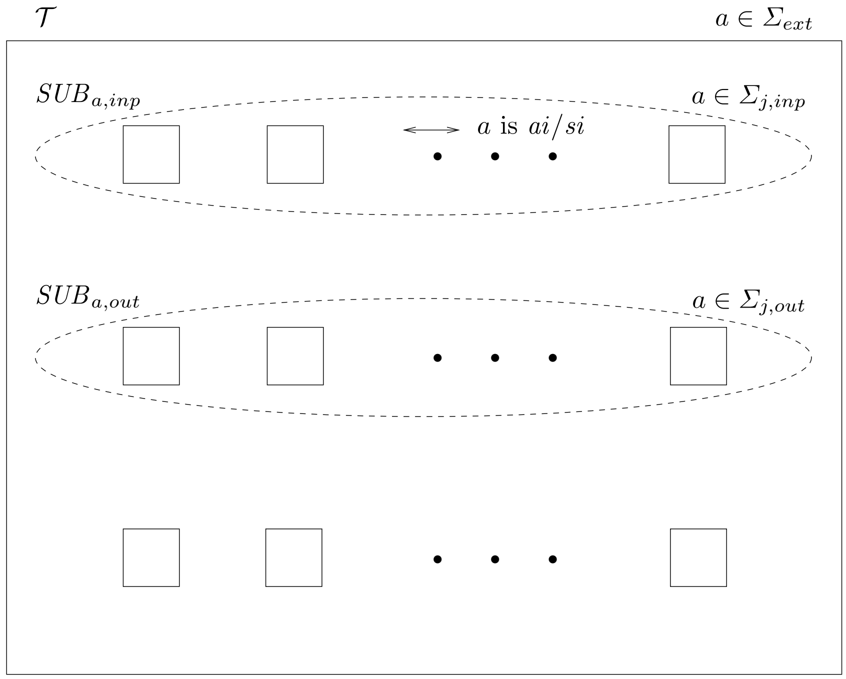


Fig. 5.5. A team automaton $\mathcal{T}$ with a *sipp*/*wipp* action a.

Example 5.3.6. (Example 5.3.2 continued) Actions a and b both are *sopp* as well as *wopp* in $\mathcal{T}$. This can be concluded from the fact that we already know from Example 4.4.4 that actions a and b both are *ai* in the output subteam $\mathcal{T}_1 = \mathcal{T}_{\{1,2\}}$ of $\mathcal{T}$. It is easy to verify that actions a and b both are also *sipp* as well as *wipp* in $\mathcal{T}$. □

5.3.2 Master-Slave

We now define synchronizations *between* the input and output subteams of an external action a. Here the idea is that input actions ("slaves") are driven by output actions ("masters"). This means that if a is an output action, then its input counterpart can never take place without being triggered (i.e. the slave never proceeds on its own). Consequently, the input subteam of an output action a cannot execute a unless a is also executed as an output action (by its output subteam). It is however possible that a is executed as an output action without its simultaneous execution as an input action. We say that a is *master-slave* if it is an output action and its output subteam participates in every a-transition of $\mathcal{T}$.

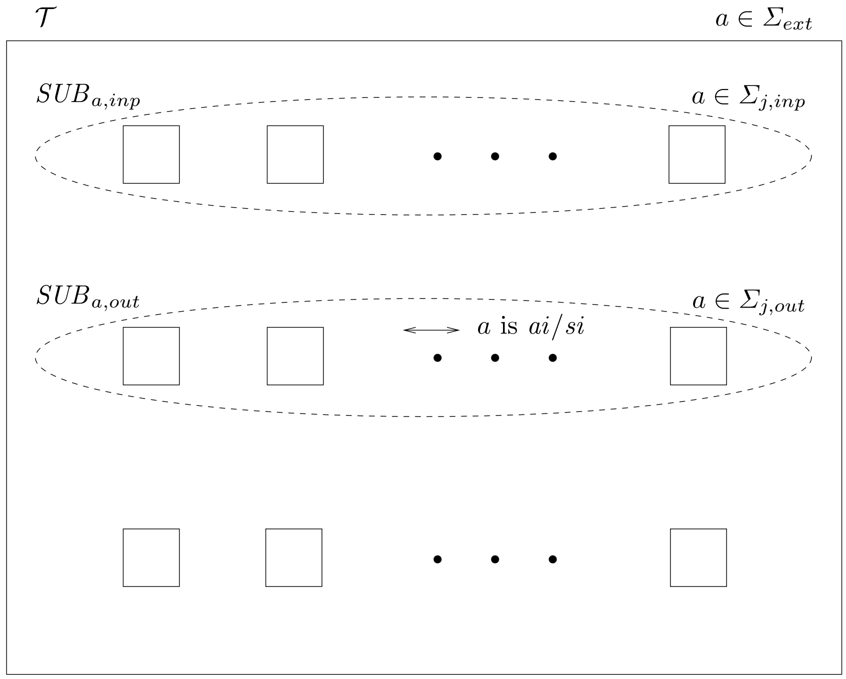


Fig. 5.6. A team automaton $\mathcal{T}$ with a *sopp*/*wopp* action a.

In addition one could require that a in its role of input action *has to* synchronize with a as an output action (i.e. the slave has to follow the master). Since the obligation of the slave to follow the master may again be formulated in two different ways, we obtain notions of *strong* and *weak master-slave* actions. When guided by the *ai* principle, we get a strong notion of master-slave synchronization, while the *si* principle leads to a weak notion of master-slave synchronization. We say that a is *strong master-slave* if it is master-slave and its input subteam moreover participates in every a-transition of $\mathcal{T}$. We say that a is *weak master-slave* if it is master-slave and its input subteam moreover participates in every a-transition of $\mathcal{T}$ *whenever it can*.

Definition 5.3.7. *Let $a \in \Sigma_{out}$, and let $J = \mathcal{I}_{a,out}$ and $K = \mathcal{I}_{a,inp}$. Then*

(1) the set of master-slave *(ms for short) actions of $\mathcal{T}$ is denoted by $MS(\mathcal{T})$ and is defined as*

$$MS(\mathcal{T}) = \{a \in \Sigma_{out} \mid proj_J{}^{[2]}(\delta_a) \subseteq (\delta_J)_a\},$$

(2) the set of strong master-slave *(sms for short) actions of $\mathcal{T}$ is denoted by $SMS(\mathcal{T})$ and is defined as*

$$SMS(\mathcal{T}) = \{a \in \Sigma_{out} \mid a \in MS(\mathcal{T}) \wedge ([K \neq \varnothing] \Rightarrow [\mathit{proj}_K{}^{[2]}(\delta_a) \subseteq (\delta_K)_a])\}, \textit{ and}$$

(3) the set of weak master-slave *(wms for short) actions of $\mathcal{T}$ is denoted by $WMS(\mathcal{T})$ and is defined as*

$$WMS(\mathcal{T}) = \{a \in \Sigma_{out} \mid a \in MS(\mathcal{T}) \wedge ([K \neq \varnothing] \Rightarrow [((q,q') \in \delta_a \wedge a \ en_{SUB_K} \ \mathit{proj}_K(q)) \Rightarrow (\mathit{proj}_K{}^{[2]}(q,q') \in (\delta_K)_a)])\}. \ \square$$

For a to be *ms*, we require it to occur at least once as an output action ($\mathcal{I}_{a,out} \neq \varnothing$) — i.e. a can act as a master. Otherwise we could have slaves without a master. A master without slaves is allowed: $\mathcal{I}_{a,out} \neq \varnothing$ and $\mathcal{I}_{a,inp} = \varnothing$. In that case a trivially is *sms* and *wms*, since there are no slaves that do not follow the master.

Since the definition of a being *ms* in $\mathcal{T}$ guarantees that the output subteam of a is actively involved in every a-transition of $\mathcal{T}$, it follows immediately from Definition 4.1.6 that the a-transitions of the output subteam of a are precisely the projections of the a-transitions of $\mathcal{T}$ on the output domain of a. Similarly, in case a is *sms* we have in addition that the a-transitions of the input subteam of a are precisely the projections of the a-transitions of $\mathcal{T}$ on the input domain of a.

Theorem 5.3.8. *Let $J = \mathcal{I}_{a,out}$ and let $K = \mathcal{I}_{a,inp}$. Then*

(1) if $a \in MS(\mathcal{T})$, then $\mathit{proj}_J{}^{[2]}(\delta_a) = (\delta_J)_a$, and

(2) if $a \in SMS(\mathcal{T})$, then $\mathit{proj}_K{}^{[2]}(\delta_a) = (\delta_K)_a$.

Proof. (1) By Definition 4.1.6 we have $(\delta_J)_a = \mathrm{proj}_J{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_j \mid j \in J\})$. Since $a \in MS(\mathcal{T})$ we have $\mathrm{proj}_J{}^{[2]}(\delta_a) \subseteq (\delta_J)_a$, for $J = \mathcal{I}_{a,out}$. Hence in this case $(\delta_J)_a = \mathrm{proj}_J{}^{[2]}(\delta_a)$.

(2) Analogous. Note that if $K = \varnothing$, then $\mathrm{proj}_K{}^{[2]}(\delta_a) = \varnothing = (\varnothing)_a$. $\square$

Note that if a is *wms*, then there may be a-transitions in $\mathcal{T}$ in which the input subteam — even when it is not trivial — is not actively involved. In those cases a is executed as an output action by $\mathcal{T}$ without the simultaneous execution of a as an input action.

Note that in Definition 5.3.7 input subteams and output subteams are treated as given entities (black boxes). Clearly, one can combine the master-slave types of synchronization with additional requirements on the synchronizations taking place within the subteams. One might, e.g., prescribe a master-slave type of synchronization on an action a that is in addition input peer-to-peer, in which case *all* component automata with a as an input action have to follow the output action. We will come back to this later.

Example 5.3.9. (Example 5.3.5 continued) If for an external action a of $\mathcal{T}$, $SUB_{a,out}$ is involved in all a-transitions of $\mathcal{T}$, then a is an *ms* action. If $SUB_{a,inp}$ moreover "has to" participate in every a-transition of $\mathcal{T}$, then a is an *sms* or *wms* action in $\mathcal{T}$. The idea of (strong or weak) types of master-slave synchronization between input and output subteams, is sketched in Figure 5.7. □

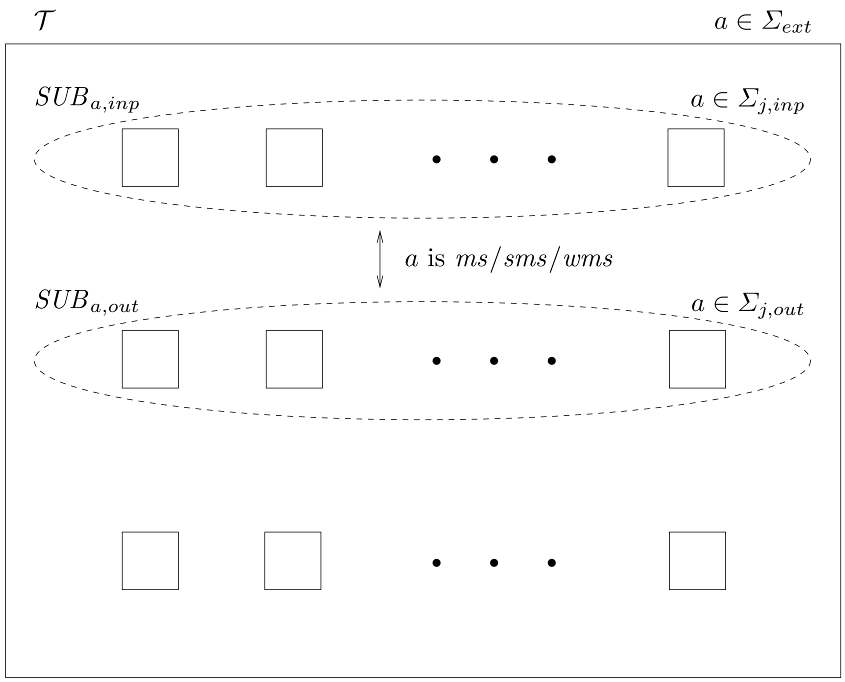


Fig. 5.7. A team automaton $\mathcal{T}$ with a *ms*/*sms*/*wms* action a.

We thus note that whereas peer-to-peer types of synchronization are defined *within* subteams, master-slave types of synchronization are defined *between* input and output subteams.

Next we give a more elaborate example in which we apply the various types of synchronization introduced in this chapter so far to one of our running examples.

Example 5.3.10. (Example 5.3.6 continued) In this example we show that the car $\mathcal{T}$ is actually a two-wheel drive. Recall that we assume a maximal interpretation of the involvement of component automata in loops.

Actions a and b are both *sms* in $\mathcal{T}$. For a this can be concluded from the fact that $\mathrm{proj}_{\{1\}}{}^{[2]}(\delta_a) = \{((s_1,s_2),(t_1,t_2)),((t_1,t_2),(t_1,t_2))\} = (\delta_{\{1\}})_a$ and $\mathrm{proj}_{\{2,3\}}{}^{[2]}(\delta_a) = \{((s_3,s_4),(t_3,t_4)),((t_3,t_4),(t_3,t_4))\} = (\delta_{\{2,3\}})_a$, thus satisfying (1) and (2) of Definition 5.3.7. For b one can verify this in a similar fashion. We thus conclude that $\mathcal{T}$ models a two-wheel drive, in the sense that one axle (the input subteam of a and b) only turns and halts as a reaction to the other axle (the output subteam of a and b). Hence the former axle is the "slave" of the latter axle. $\square$

5.3.3 A Case Study

In [Ell97] a simple example was presented to illustrate the concept of peer-to-peer and master-slave types of synchronization within team automata. In this subsection we give this example from [Ell97] a rigorous treatment in our formal team automata framework.

Example 5.3.11. Consider the three component automata depicted in Figure 5.8. They are formally defined by $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$, where for $i \in [3]$,

$Q_i = \{q_i, q_i'\}$,
$\Sigma_{1,inp} = \Sigma_{2,inp} = \Sigma_{3,out} = \varnothing$,
$\Sigma_{1,out} = \Sigma_{2,out} = \Sigma_{3,inp} = \{b\}$,
$\Sigma_{i,int} = \{a_i, a_i'\}$, with all a_i and a_i' distinct symbols different from b,
$\delta_{i,b} = \{(q_i, q_i')\}$,
$\delta_{j,a_j} = \{(q_j, q_j')\}$ and $\delta_{j,a_j'} = \{(q_j', q_j)\}$, for $j \in [2]$,
$\delta_{3,a_3} = \{(q_3, q_3)\}$ and $\delta_{3,a_3'} = \{(q_3', q_3')\}$, and
$I_i = \{q_i\}$.

Hence $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ is a composable system.

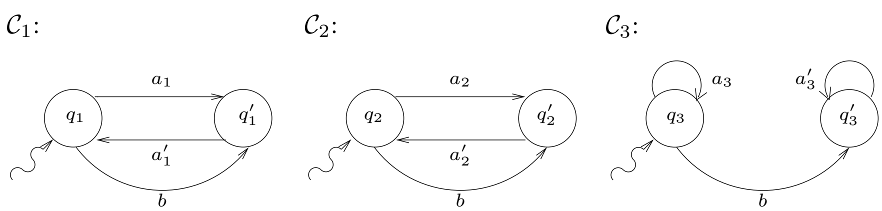


Fig. 5.8. Component automata $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$.

Two slightly different team automata $\mathcal{T}$ and $\mathcal{T}'$ over this composable system are defined next. All parameters of these team automata, except for the set of labeled transitions, are predetermined by $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$. In fact, only the b-transitions can be varied as all the other actions are internal. The first team automaton ($\mathcal{T}$) is the one spelled out in [Ell97], whereas the second one ($\mathcal{T}'$) is the one discussed in the text in [Ell97].

Let $\mathcal{T} = (\prod_{i\in[3]} Q_i, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, \{(q_1, q_2, q_3)\})$ and let $\mathcal{T}' = (\prod_{i\in[3]} Q_i, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta', \{(q_1, q_2, q_3)\})$, where

$\Sigma_{inp} = \varnothing$,
$\Sigma_{out} = \{b\}$,
$\Sigma_{int} = \{a_1, a_1', a_2, a_2', a_3, a_3'\}$, and
δ and δ' are defined by

$\delta_a = \delta_a' = \Delta_a(\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\})$, for each $a \in \{a_1, a_1', a_2, a_2', a_3, a_3'\}$,
$\delta_b = \{((q_1, q_2, q_3), (q_1', q_2', q_3'))\}$, and
$\delta_b' = \{((q_1, q_2, q_3), (q_1', q_2', q_3')), ((q_1, q_2, q_3'), (q_1', q_2', q_3'))\}$.

Hence in $\mathcal{T}$ there is only one b-transition that can take place. It involves all three component automata and requires the j-th component to be in state q_j, for each $j \in [3]$. This transition is thus a simultaneous execution of b by all three component automata. In $\mathcal{T}'$, however, next to this b-transition just described, there is another b-transition that can take place and it involves only the first two component automata while the third component automaton is in state q_3' (in which b is not enabled). Hence this transition is a simultaneous execution of b by the first two component automata only. Both these team automata are depicted in Figure 5.9: $\mathcal{T}'$ contains all the depicted transitions, whereas $\mathcal{T}$ is obtained by ignoring the "dashed" transition $((q_1, q_2, q_3'), b, (q_1', q_2', q_3'))$.

It is easy to check that $\mathit{Free}(\mathcal{T}) = \mathit{Free}(\mathcal{T}') = \mathit{AI}(\mathcal{T}') = \Sigma_{int}$ and $\mathit{AI}(\mathcal{T}) = \mathit{SI}(\mathcal{T}) = \mathit{SI}(\mathcal{T}') = \Sigma$. Thus b is both *si* and *ai* in $\mathcal{T}$, while b is *si* but not *ai* in $\mathcal{T}'$. This is because $\mathcal{T}'$ has a b-transition in which $\mathcal{C}_3$ does not participate, even though $\mathcal{C}_3$ contains b in its (input) alphabet.

Note that in $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ the input domain $\mathcal{I}_{b,inp}$ of b is $\{3\}$ and the output domain $\mathcal{I}_{b,out}$ of b is $\{1, 2\}$. The subteams of $\mathcal{T}$ and $\mathcal{T}'$ determined by $\{1, 2\}$ are the same: $\mathit{SUB}_{\{1,2\}}(\mathcal{T}) = \mathit{SUB}_{\{1,2\}}(\mathcal{T}')$. This is because $\mathrm{proj}_{\{1,2\}}{}^{[2]}(\delta_c) = \mathrm{proj}_{\{1,2\}}{}^{[2]}(\delta_c')$, for each $c \in \{a_1, a_1', a_2, a_2', b\}$. Also $\mathit{SUB}_{\{3\}}(\mathcal{T}) = \mathit{SUB}_{\{3\}}(\mathcal{T}')$, since $\mathrm{proj}_{\{3\}}{}^{[2]}(\delta_c) = \mathrm{proj}_{\{3\}}{}^{[2]}(\delta_c')$, for each $c \in \{a_3, a_3'\}$, and $\mathrm{proj}_{\{3\}}{}^{[2]}(\delta_b) \cap \Delta_b(\{\mathcal{C}_3\}) = \mathrm{proj}_{\{3\}}{}^{[2]}(\delta_b') \cap \Delta_b(\{\mathcal{C}_3\}) = \{((q_3), (q_3'))\}$.

Since b is *ai* in $\mathcal{T}$, Lemma 4.7.1(2) implies that b is also *ai* in both $\mathit{SUB}_{\{1,2\}}(\mathcal{T}) = \mathit{SUB}_{\{1,2\}}(\mathcal{T}')$ and $\mathit{SUB}_{\{3\}}(\mathcal{T}) = \mathit{SUB}_{\{3\}}(\mathcal{T}')$. From this it follows that b is both *sopp* and *sipp* in $\mathcal{T}$ as well as in $\mathcal{T}'$.

Moreover, action b is *ms* in both $\mathcal{T}$ and $\mathcal{T}'$ since we have $\mathrm{proj}_{\{1,2\}}{}^{[2]}(\delta_b) = \mathrm{proj}_{\{1,2\}}{}^{[2]}(\delta_b') = \{((q_1, q_2), (q_1', q_2'))\} \subseteq \{((q_1, q_2), (q_1', q_2'))\} = (\delta_{\{1,2\}})_b =$

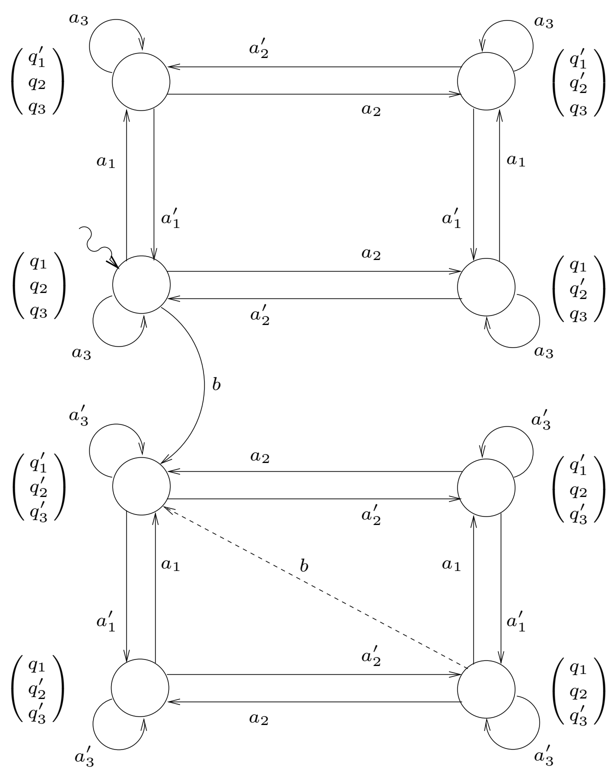


Fig. 5.9. Team automata $\mathcal{T}$ and $\mathcal{T}'$.

$(\delta'_{\{1,2\}})_b$, i.e. the output subteam of b participates in every b-transition of the team automata. In fact, b is even *sms* in $\mathcal{T}$ as b is *ms* in $\mathcal{T}$ and $\text{proj}_{\{3\}}{}^{[2]}(\delta_b) = \{(q_3, q'_3)\} \subseteq \{(q_3, q'_3)\} = (\delta_{\{3\}})_b$, i.e. also the input subteam of b participates in every b-transition of $\mathcal{T}$. It is clear that b is also *wms* in $\mathcal{T}$. However, $\text{proj}_{\{3\}}{}^{[2]}(\delta'_b) = \{((q_3),(q'_3)),((q'_3),(q'_3))\} \nsubseteq \{((q_3),(q'_3))\} = (\delta'_{\{3\}})_b$ and b is thus not *sms* in $\mathcal{T}'$. Since q_3 is the only state of $\mathcal{C}_3$ at which b is enabled in $\mathcal{C}_3$ we do have that b is *wms* in $\mathcal{T}'$.

The fact that $\mathcal{T}$ does not allow an output action b to take place without a “slave” input action b leads to b being *sms* in $\mathcal{T}$. In $\mathcal{T}'$, however, b is *wms* since the input action b follows the “master” output action b only when enabled.

To understand that despite the similarities this subtle difference — due to the distinction between *ai* and *si* — may lead to different externally observable behaviors of $\mathcal{T}$ and $\mathcal{T}'$, it is sufficient to show that $ba'_1a'_2b \in \mathbf{B}^{\Sigma}_{\mathcal{T}'}$ while no word with two b's is contained in $\mathbf{B}^{\Sigma}_{\mathcal{T}}$. The computation $(q_1,q_2,q_3)b(q'_1,q'_2,q'_3)a'_1(q_1,q'_2,q'_3)a'_2(q_1,q_2,q'_3)b(q'_1,q'_2,q'_3) \in \mathbf{C}_{\mathcal{T}'}$ proves that $ba'_1a'_2b \in \mathbf{B}^{\Sigma}_{\mathcal{T}'}$, whereas in δ the execution of b from the initial state (q_1, q_2, q_3) always leads to (q'_1, q'_2, q'_3), after which (q_1, q_2, q_3) — the only state from which b can be executed — has become unreachable. $\square$

5.3.4 Peer-to-Peer and Master-Slave

We continue our comparison of the various types of synchronization started in Subsection 4.4.4 by extending our study to the types of synchronization introduced in this section.

First we revisit the synchronizations introduced in Section 4.4. This time, however, we deal with team automata rather than synchronized automata and we thus have a distribution of the alphabet of actions into input, output, and internal actions. We immediately note that if a is an internal action of one of the component automata of a team automaton $\mathcal{T}$, then it is not an action of any other component automaton of $\mathcal{T}$, in which case a thus trivially is *free*, *ai*, and *si* in $\mathcal{T}$.

Lemma 5.3.12. $\Sigma_{int} \subseteq \mathit{Free}(\mathcal{T}) \cap \mathit{AI}(\mathcal{T})$.

Proof. Let $a \in \Sigma_{int}$. From Definition 5.1.4 it follows that for all $(q, q') \in \delta_a$ there exists a unique $i \in \mathcal{I}$ such that $(\mathrm{proj}_i(q), a, \mathrm{proj}_i(q')) \in \delta_i$ and, moreover, $a \notin \bigcup_{j \in \mathcal{I} \setminus \{i\}} \Sigma_j$. Hence a trivially is *free*, *ai*, and *si*. $\square$

We continue our investigation by involving also the synchronizations introduced in Section 5.3. We begin by comparing the various types of peer-to-peer (master-slave) synchronization among each other.

Definition 5.3.4 and Lemma 4.4.7 directly imply that actions that are *sipp* (*sopp*) are also *wipp* (*wopp*).

Lemma 5.3.13. *(1)* $\mathit{SIPP}(\mathcal{T}) \subseteq \mathit{WIPP}(\mathcal{T})$ *and*

(2) $\mathit{SOPP}(\mathcal{T}) \subseteq \mathit{WOPP}(\mathcal{T})$. $\square$

From Example 4.4.8 we immediately conclude that the inclusions of this lemma in general do not hold the other way around.

From Definition 5.3.7 we immediately obtain that the fact that an action is *sms* implies that it is *wms*, which in its turn implies that it is *ms*.

Lemma 5.3.14. $\mathit{SMS}(\mathcal{T}) \subseteq \mathit{WMS}(\mathcal{T}) \subseteq \mathit{MS}(\mathcal{T})$. $\square$

In Example 5.3.11 we have seen an example of a synchronization that is *wms* but not *sms*. This implies that also the inclusion of this lemma in general does not hold the other way around.

We now continue our investigation by comparing the various types of peer-to-peer (master-slave) synchronizations with the types of synchronization introduced in Section 4.4.

First we consider the types of peer-to-peer synchronization. Recall that $\Sigma_{out} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$, whereas Σ_{inp} need not equal $\bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$.

Theorem 5.3.15. *(1)* $(\bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}) \cap AI(\mathcal{T}) \subseteq SIPP(\mathcal{T})$,

(2) $(\bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}) \cap SI(\mathcal{T}) \subseteq WIPP(\mathcal{T})$,

(3) $\Sigma_{out} \cap AI(\mathcal{T}) \subseteq SOPP(\mathcal{T})$, *and*

(4) $\Sigma_{out} \cap SI(\mathcal{T}) \subseteq WOPP(\mathcal{T})$.

Proof. (1) Let $a \in (\bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}) \cap AI(\mathcal{T})$. According to Definition 5.3.4(1) it remains to prove that $a \in AI(SUB_{a,inp})$. However, $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$ implies that $\mathcal{I}_{a,inp} \neq \varnothing$ and since $a \in AI(\mathcal{T})$, it thus follows directly from Lemma 4.7.1(2) that $a \in AI(SUB_{a,inp})$.

(2-4) Analogous. □

In the following example we show that in general none of the inclusions of this theorem holds also the other way around.

Example 5.3.16. (Example 4.4.8 continued) We turn automata $\mathcal{A}_1$ and $\mathcal{A}_2$ into component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, each with input action a. This is done in the obvious way, viz. $\mathcal{C}_1 = (\{q, q'\}, (\{a\}, \varnothing, \varnothing), \{(q, a, q')\}, \{q\})$ and $\mathcal{C}_2 = (\{r, r'\}, (\{a\}, \varnothing, \varnothing), \{(r, a, r')\}, \{r\})$. Note that $\text{und}(\mathcal{C}_1) = \mathcal{A}_1$ and $\text{und}(\mathcal{C}_2) = \mathcal{A}_2$ are depicted in Figure 4.10.

Now consider the team automaton $\widehat{\mathcal{T}}^1 = (\{(q, r), (q, r'), (q', r), (q', r')\}, (\{a\}, \varnothing, \varnothing), \delta^1, \{(q, r)\})$, where we recall that $\delta^1 = \{((q, r), a, (q, r')), ((q, r), a, (q', r'))\}$. Then it is clear that input action a is not *si* and thus neither *ai*. However, in $SUB_{\{2\}}(\widehat{\mathcal{T}}^1)$ — which is essentially a copy of $\mathcal{C}_2$ — action a trivially is *sipp* and *wipp*.

In an analogous way we can show that in general neither of the inclusions stated in Theorem 5.3.15(3,4) holds the other way around as well. □

Next we consider the types of master-slave synchronization.

Theorem 5.3.17. $\Sigma_{out} \cap AI(\mathcal{T}) \subseteq MS(\mathcal{T})$.

Proof. Let $a \in \Sigma_{out} \cap AI(\mathcal{T})$ and let $(q, q') \in \delta_a$. Then for all $j \in \mathcal{I}_{a,out}$, we have that $\text{proj}_j{}^{[2]}(q, q') \in \delta_{j,a}$. This implies that it must be the case that $\text{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \subseteq (\delta_{\mathcal{I}_{a,out}})_a$ and thus $a \in MS(\mathcal{T})$. □

In the following example we show that in general the inclusion of this theorem does not hold also the other way around.

Example 5.3.18. Consider the composable system $\{\mathcal{C}_1, \mathcal{C}_2\}$ consisting of component automata $\mathcal{C}_i = (\{q_i, q_i'\}, (\varnothing, \{a\}, \varnothing), \{(q_i, a, q_i')\}, \{q_i\})$, with $i \in [2]$. It is depicted in Figure 5.10(a).

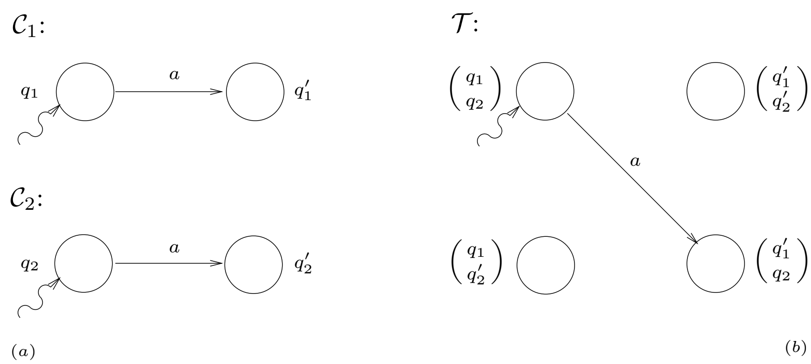


Fig. 5.10. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, and team automaton $\mathcal{T}$.

Now consider team automaton $\mathcal{T} = (\{(q_1, q_2), (q_1', q_2), (q_1, q_2'), (q_1', q_2')\}, (\varnothing, \{a\}, \varnothing), \{((q_1, q_2), a, (q_1', q_2))\}, \{(q_1, q_2)\})$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$, depicted in Figure 5.10(b).

Clearly $\mathcal{I}_{a,out}(\{\mathcal{C}_1, \mathcal{C}_2\}) = \{1, 2\}$. Hence a trivially is *ms* (*sms*, *wms*) in $\mathcal{T}$, but a is not *ai* in $\mathcal{T}$ since $\mathcal{C}_2$ does not participate in the a-transition of $\mathcal{T}$ even though it has a in its alphabet. □

The preceding two theorems immediately imply the following result.

Corollary 5.3.19. $\Sigma_{out} \cap AI(\mathcal{T}) \subseteq SOPP(\mathcal{T}) \cap MS(\mathcal{T})$. □

Finally we involve also *sms* and *wms* actions.

Theorem 5.3.20. *If $\Sigma_{out} \subseteq AI(\mathcal{T})$, then $MS(\mathcal{T}) = SMS(\mathcal{T}) = WMS(\mathcal{T})$.*

Proof. Let $\Sigma_{out} \subseteq AI(\mathcal{T})$. Now let $a \in MS(\mathcal{T})$. Then by Definition 5.3.7(1), $a \in \Sigma_{out}$ and thus also $a \in AI(\mathcal{T})$. We distinguish two cases.
If there does not exist a $j \in \mathcal{I}$ such that $a \in \Sigma_{j,inp}$, then $\mathcal{I}_{a,inp} = \varnothing$ and thus trivially $a \in SMS(\mathcal{T})$.
If there exist a $j \in \mathcal{I}$ such that $a \in \Sigma_{j,inp}$, then $\mathcal{I}_{a,inp} \neq \varnothing$ and, because a is *ai*, $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(\delta_a) \subseteq (\delta_{\mathcal{I}_{a,inp}})_a$. Hence $a \in SMS(\mathcal{T})$.

In both cases we thus obtain that $a \in SMS(\mathcal{T})$. Hence $MS(\mathcal{T}) \subseteq SMS(\mathcal{T})$ and since, by Lemma 5.3.14, $SMS(\mathcal{T}) \subseteq WMS(\mathcal{T}) \subseteq MS(\mathcal{T})$ the equality follows. □

5.4 Predicates of Synchronizations

In the preceding sections of this chapter we have presented our team automata framework. We have seen that team automata over composable systems are themselves component automata that can be used in further constructions of team automata. Team automata can thus be used as building blocks. We have analyzed the transition relations of team automata in order to determine whether or not they satisfy the conditions inherent to certain specific types of synchronization modeling collaboration between system components. However, we have seen that these conditions in general do not lead to uniquely defined team automata.

To make the model of team automata of any use, e.g. in the early phases of system design, it is necessary to be able to unambiguously construct a team automaton according to the specification of the required type of synchronization. Given a composable system and certain conditions to be satisfied by the synchronizations, we want to construct the unique team automaton over this composable system. This is done in very much the same way as we constructed the *maximal-free* (*maximal-ai*, *maximal-si*) synchronized automata of Section 4.5, viz. by defining predicates of synchronization. Since for an internal action the transition relation is by definition equal to its complete transition space in $\mathcal{S}$, we need to choose predicates only for all external actions. Once we do so, the team automaton over $\mathcal{S}$ defined by these predicates is unique.

Based on Definition 4.5.1, this is formalized as follows.

Definition 5.4.1. *Let $\mathcal{R}_a(\mathcal{S}) \subseteq \Delta_a(\mathcal{S})$, for all $a \in \Sigma_{ext}$, and let $\mathcal{R}_a(\mathcal{S}) = \Delta_a(\mathcal{S})$, for all $a \in \Sigma_{int}$. Let $\mathcal{R} = \{\mathcal{R}_a(\mathcal{S}) \mid a \in \Sigma\}$. Then $\mathcal{T}$ is* the $\mathcal{R}$-team automaton *over $\mathcal{S}$ if for all $a \in \Sigma$,*

$$\delta_a = \mathcal{R}_a(\mathcal{S}). \qquad \square$$

In Section 4.5 we have seen that each of the predicates $\mathcal{R}_a^{free}(\mathcal{S})$, $\mathcal{R}_a^{ai}(\mathcal{S})$, and $\mathcal{R}_a^{si}(\mathcal{S})$ defines the largest transition relation in $\Delta_a(\mathcal{S})$ in which an action a is *free*, *ai*, and *si*, respectively.

As an immediate corollary of Theorem 4.5.5 we obtain that in case of an internal action, each such a predicate equals the *no-constraints* predicate, i.e. its complete transition space in $\mathcal{S}$.

Theorem 5.4.2. *Let $a \in \Sigma_{int}$. Then*

$$\Delta_a(\mathcal{S}) = \mathcal{R}_a^{no}(\mathcal{S}) = \mathcal{R}_a^{syn}(\mathcal{S}), \text{ for all } syn \in \{free, ai, si\}. \qquad \square$$

The generic setup of Definition 5.4.1 now allows us to define three specific team automata as an extension of Definition 4.5.4.

Definition 5.4.3. *Let syn* $\in \{free, ai, si\}$*. Then*

the $\{\mathcal{R}_a^{syn}(\mathcal{S}) \mid a \in \Sigma\}$*-team automaton over* $\mathcal{S}$ *is called the* maximal-syn team automaton *(over* $\mathcal{S}$*).* □

We now consider the constraints relating to the types of synchronization defined in Section 5.3. This will allow us to define more types of team automata than those of Definition 5.4.3. We define the predicates of synchronization without any reference to a team automaton, its subteams, and its transition relation.

We begin by considering the peer-to-peer types of synchronization. In this case we have to distinguish between the input and output role an external action a may have in $\mathcal{S}$. The predicates thus have to refer to the input and output domains of a in $\mathcal{S}$. Moreover, we have to distinguish between strong (ai) and weak (si) types of synchronization. This leads to four predicates, each of which includes all and only those transitions from $\Delta_a(\mathcal{S})$ in which all component automata given by the input or output domain, respectively, are forced (in the strong or in the weak sense) to participate.

Recall that, for an external action a, $\mathcal{I}_{a,inp}(\mathcal{S}) = \{i \in \mathcal{I} \mid a \in \Sigma_{i,inp}\}$ is the input domain of a in $\mathcal{S}$ and $\mathcal{I}_{a,out}(\mathcal{S}) = \{i \in \mathcal{I} \mid a \in \Sigma_{i,out}\}$ is the output domain of a in $\mathcal{S}$. As before, we may simply write $\mathcal{I}_{a,inp}$ and $\mathcal{I}_{a,out}$, since $\mathcal{S}$ has been fixed.

First we focus on input actions.

Definition 5.4.4. *Let* $a \in \Sigma$ *and let* $\mathcal{S}_{a,inp} = \{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\}$*. Then*

(1) the predicate is-sipp *in* $\mathcal{S}$ *for* a *is denoted by* $\mathcal{R}_a^{sipp}(\mathcal{S})$ *and is defined as*

if $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$*, then*

$$\mathcal{R}_a^{sipp}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathit{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,inp}) \Rightarrow \mathit{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \mathcal{R}_a^{ai}(\mathcal{S}_{a,inp})\},$$

otherwise

$$\mathcal{R}_a^{sipp}(\mathcal{S}) = \Delta_a(\mathcal{S}), \text{ and}$$

(2) the predicate is-wipp *in* $\mathcal{S}$ *for* a *is denoted by* $\mathcal{R}_a^{wipp}(\mathcal{S})$ *and is defined as*

if $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$, *then*

$$\mathcal{R}_a^{wipp}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,inp}) \Rightarrow proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \mathcal{R}_a^{si}(\mathcal{S}_{a,inp})\},$$

otherwise

$$\mathcal{R}_a^{wipp}(\mathcal{S}) = \Delta_a(\mathcal{S}). \qquad \square$$

Next we focus on output actions.

Definition 5.4.5. *Let $a \in \Sigma$ and let $\mathcal{S}_{a,out} = \{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\}$. Then*

(1) the predicate is-sopp *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{sopp}(\mathcal{S})$ and is defined as*

if $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$, *then*

$$\mathcal{R}_a^{sopp}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,out}) \Rightarrow proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \mathcal{R}_a^{ai}(\mathcal{S}_{a,out})\},$$

otherwise

$$\mathcal{R}_a^{sopp}(\mathcal{S}) = \Delta_a(\mathcal{S}), \text{ and}$$

(2) the predicate is-wopp *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{wopp}(\mathcal{S})$ and is defined as*

if $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$, *then*

$$\mathcal{R}_a^{wopp}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,out}) \Rightarrow proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \mathcal{R}_a^{si}(\mathcal{S}_{a,out})\},$$

otherwise

$$\mathcal{R}_a^{wopp}(\mathcal{S}) = \Delta_a(\mathcal{S}). \qquad \square$$

One should recall at this point that we are not discussing the properties of a given team automaton over $\mathcal{S}$, with a fixed transition relation determining the transitions in the input and output subteams of an external action a. Thus, in Definitions 5.4.4 and 5.4.5, we relate to the complete transition spaces of a in the respective "subsystems" determined by the input and output domain of a. Each predicate includes all and only those transitions from $\Delta_a(\mathcal{S})$, for which all component automata given by the input or output domain, respectively, are forced (in the weak or in the strong sense) to participate in the execution of a by any of these component automata.

As the next result shows, the predicates of Definitions 5.4.4 and 5.4.5 describe the maximal sets of a-transitions satisfying the given constraint. Recall that $\Sigma_{out} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$.

Theorem 5.4.6. *Let $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$. Then*

(1) $a \in SIPP(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{sipp}(\mathcal{S})$, and

(2) $a \in WIPP(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{wipp}(\mathcal{S})$.

Let $a \in \Sigma_{out}$. Then

(3) $a \in SOPP(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{sopp}(\mathcal{S})$, and

(4) $a \in WOPP(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{wopp}(\mathcal{S})$.

Proof. (1) (Only if) Let $a \in SIPP(\mathcal{T})$. Hence according to Definition 5.3.4(1) we have $a \in AI(SUB_{a,inp})$, i.e. a is *ai* in the subteam of $\mathcal{T}$ determined by the input domain of a. According to Definition 4.1.6 the a-transitions of this subteam are $(\delta_{\mathcal{I}_{a,inp}})_a = \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Now, by Theorem 4.5.3(2), $a \in AI(SUB_{a,inp})$ implies that $(\delta_{\mathcal{I}_{a,inp}})_a \subseteq \mathcal{R}_a^{ai}(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Hence for all $(q,q') \in \delta_a$, whenever $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$, then $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \mathcal{R}_a^{ai}(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Consequently, according to Definition 5.4.4(1), $\delta_a \subseteq \mathcal{R}_a^{sipp}(\mathcal{S})$.

(If) Let $\delta_a \subseteq \mathcal{R}_a^{sipp}(\mathcal{S})$. By Definition 5.3.4(1) we now have to prove that $a \in AI(SUB_{a,inp})$. Since $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$, we know that $\mathcal{I}_{a,inp} \neq \varnothing$. Hence consider an arbitrary pair $(p,p') \in (\delta_{\mathcal{I}_{a,inp}})_a$. Since $(p,p') \in (\delta_{\mathcal{I}_{a,inp}})_a = \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$ there is a $(q,q') \in \delta_a \subseteq \Delta_a(\mathcal{S})$ for which $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') = (p,p')$. From $\delta_a \subseteq \mathcal{R}_a^{sipp}(\mathcal{S})$ we infer that $(p,p') \in \mathcal{R}_a^{ai}(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Hence $(\delta_{\mathcal{I}_{a,inp}})_a \subseteq \mathcal{R}_a^{ai}(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$ and thus, by Theorem 4.5.3(2), $a \in AI(SUB_{a,inp})$.

(2-4) Analogous. □

Now we turn to the master-slave types of synchronization. As in the case of the peer-to-peer predicates, we have to distinguish between the input and the output role of actions. This time, however, the predicates describe synchronizations *between* the component automata from the input domain and those from the output domain.

The *is-ms* predicate for an external action a includes all and only those a-transitions in which a appears at least once in its output role. For the predicates *is-sms* and *is-wms* in $\mathcal{S}$, there is the additional requirement that a should also be executed by the component automata from its input domain. In the strong case, this obligation is strict in the sense that if the input domain of a is not empty, then always at least one component automaton from the input domain of a participates in every a-transition included in the predicate. In the weak case, this obligation has to be met only when at least one component automaton from the input domain of a is ready to execute a.

Definition 5.4.7. *Let $a \in \Sigma$, let $\mathcal{S}_{a,inp} = \{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\}$, and let $\mathcal{S}_{a,out} = \{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\}$. Then*

(1) the predicate is-ms *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{ms}(\mathcal{S})$ and is defined as*

if $a \in \Sigma_{out}$, then

$$\mathcal{R}_a^{ms}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,out})\},$$

otherwise

$$\mathcal{R}_a^{ms}(\mathcal{S}) = \Delta_a(\mathcal{S}),$$

(2) the predicate is-sms *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{sms}(\mathcal{S})$ and is defined as*

if $a \in \Sigma_{out}$, then

$$\mathcal{R}_a^{sms}(\mathcal{S}) = \mathcal{R}_a^{ms}(\mathcal{S}) \cap \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathcal{I}_{a,inp} \neq \varnothing \Rightarrow proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,inp})\},$$

otherwise

$$\mathcal{R}_a^{sms}(\mathcal{S}) = \Delta_a(\mathcal{S}), \text{ and}$$

(3) the predicate is-wms *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{wms}(\mathcal{S})$ and is defined as*

if $a \in \Sigma_{out}$, then

$$\mathcal{R}_a^{wms}(\mathcal{S}) = \mathcal{R}_a^{ms}(\mathcal{S}) \cap \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathcal{I}_{a,inp} \neq \varnothing \Rightarrow [(\exists i \in \mathcal{I}_{a,inp} : a \ en_{\mathcal{C}_i} \ proj_i(q)) \Rightarrow proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,inp})]\},$$

otherwise

$$\mathcal{R}_a^{wms}(\mathcal{S}) = \Delta_a(\mathcal{S}).$$

□

The *is-ms* (*is-sms*, *is-wms*) predicate guarantees that the output action a is indeed *ms* (*sms*, *wms*) in every team automaton over $\mathcal{S}$ with that predicate for its a-transitions. The predicates *is-ms* and *is-sms*, moreover, are the largest set of a-transitions satisfying the specified constraint.

It is, however, not necessarily the case that every set of a-transitions by which a is *is-wms* is contained in the predicate *is-wms*. This difference stems from the fact that the predicate refers to component automata from the input domain of a rather than an input subteam. There is no way out and in fact the maximality principle is not applicable, because to define a subteam with transitions, a team automaton including the transition relation should have been defined already. Since a subteam only contains a selection of all possible a-transitions, it may happen that a is enabled in a component automaton of the input subteam, but not in the subteam. Thus a can be *wms* in team automaton $\mathcal{T}$ even when δ_a contains transitions in which the input subteam of a does not participate, although a is currently enabled in a component automaton of this subteam.

Theorem 5.4.8. *Let* $a \in \Sigma_{out}$. *Then*

(1) $a \in MS(\mathcal{T})$ *if and only if* $\delta_a \subseteq \mathcal{R}_a^{ms}(\mathcal{S})$,

(2) $a \in SMS(\mathcal{T})$ *if and only if* $\delta_a \subseteq \mathcal{R}_a^{sms}(\mathcal{S})$, *and*

(3) if $\delta_a \subseteq \mathcal{R}_a^{wms}(\mathcal{S})$, *then* $a \in WMS(\mathcal{T})$.

Proof. (1) (Only if) Let $a \in MS(\mathcal{T})$. Hence by Lemma 5.3.8(1) we have $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) = (\delta_{\mathcal{I}_{a,out}})_a$. By Definition 4.1.6 consequently $(\delta_{\mathcal{I}_{a,out}})_a = \mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\})$ and thus $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \subseteq \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\})$. Hence by Definition 5.4.7(1), $\delta_a \subseteq \mathcal{R}_a^{ms}(\mathcal{S})$.

(If) Let $\delta_a \subseteq \mathcal{R}_a^{ms}(\mathcal{S})$. Then by Definition 5.3.7(1) we have to prove that $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \subseteq (\delta_{\mathcal{I}_{a,out}})_a$. By Definition 4.1.6 we thus have to prove $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \subseteq \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\})$. This follows immediately from Definition 5.4.7(1).

(2) Let $a \in SMS(\mathcal{T})$. If $\mathcal{I}_{a,inp} = \varnothing$, then there is nothing to prove. Hence assume that $\mathcal{I}_{a,inp} \neq \varnothing$. As in the proof of (1), for $\mathcal{I}_{a,out}$ it is easy to prove that $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(\delta_a) \subseteq (\delta_{\mathcal{I}_{a,inp}})_a$ if and only if $\delta_a \subseteq \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})\}$. By using Definition 5.3.7(2) we thus infer that $a \in SMS(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{ms}(\mathcal{S})$ and $\delta_a \subseteq \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})\}$. Hence according to Definition 5.4.7(2) we are ready.

(3) Again there is nothing to prove whenever $\mathcal{I}_{a,inp} = \varnothing$. Hence assume that $\mathcal{I}_{a,inp} \neq \varnothing$. Let $\delta_a \subseteq \mathcal{R}_a^{wms}(\mathcal{S})$. Then by Definition 5.3.7(3) we have

to prove that whenever $(q, q') \in \delta_a$ and a en$_{SUB_{a,inp}}$ $\mathrm{proj}_{\mathcal{I}_{a,inp}}(q)$, then $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q, q') \in (\delta_{\mathcal{I}_{a,inp}})_a$. Definition 5.4.7(3) implies that for all $(q, q') \in \delta_a$, if there is an $i \in \mathcal{I}_{a,inp}$ for which a en$_{\mathcal{C}_i}$ $\mathrm{proj}_i(q)$, then $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q, q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Since a en$_{SUB_{a,inp}}$ $\mathrm{proj}_{\mathcal{I}_{a,inp}}(q)$ implies that then there is an $i \in \mathcal{I}_{a,inp}$ for which a en$_{\mathcal{C}_i}$ $\mathrm{proj}_i(q)$, we now have that if $(q, q') \in \delta_a$ and a en$_{SUB_{a,inp}}$ $\mathrm{proj}_{\mathcal{I}_{a,inp}}(q)$, then $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q, q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Now Definition 4.1.6 implies that $(\delta_{\mathcal{I}_{a,inp}})_a = \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q, q')$ and thus we are ready. $\square$

In the following example we show that, as announced before, the converse of Theorem 5.4.8(3) in general indeed does not hold.

Example 5.4.9. Let $\mathcal{C}_1 = (\{q_1, q_2\}, (\{a\}, \varnothing, \varnothing), \{(q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{C}_2 = (\{q_2, q_2'\}, (\varnothing, \{a\}, \varnothing), \{(q_2, a, q_2')\}, \{q_2\})$ be the two component automata depicted in Figure 5.11(a).

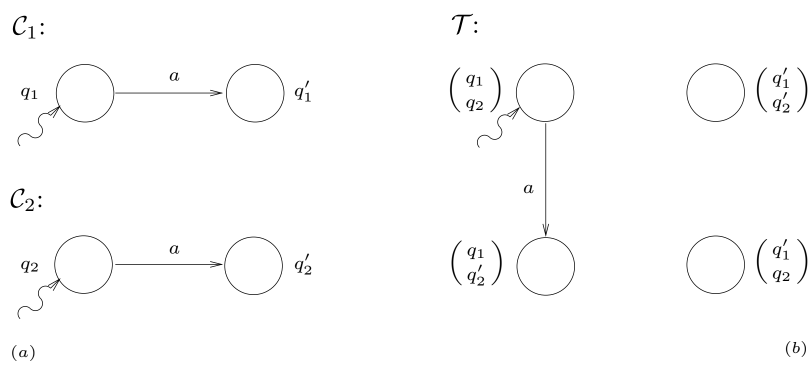


Fig. 5.11. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, and team automaton $\mathcal{T}$.

Clearly $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2\}$ is a composable system. Consider team automaton $\mathcal{T} = (\{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}, (\varnothing, \{a\}, \varnothing), \{((q_1, q_2), a, (q_1, q_2'))\}, \{(q_1, q_2)\})$ over $\mathcal{S}$. It is depicted in Figure 5.11(b). Since a is not enabled in state (q_1) of the input subteam of $\mathcal{T}$ it is trivial to see that $a \in \mathit{WMS}(\mathcal{T})$. Note however that a is enabled in state q_1 of component automaton $\mathcal{C}_1$ of the input subteam. Since this component automaton does not participate in the a-transition $((q_1, q_2), (q_1, q_2'))$ of $\mathcal{T}$, however, we have found that $((q_1, q_2), (q_1, q_2')) \in \delta_a \setminus \mathcal{R}_a^{wms}(\mathcal{S})$. $\square$

Summarizing we thus conclude that except for *wms*, each of the types of synchronization introduced in Section 5.3 — as did each of the types introduced in Section 4.4 — gives rise to a predicate that is the unique maximal representative among all transition relations satisfying the constraints implied by the type of synchronization. Consequently, we can now distinguish more specific types of team automata.

Definition 5.4.10. *Let $syn \in \{sipp, wipp, sopp, wopp, ms, sms\}$. Then*

(1) the $\{\mathcal{R}_a^{syn}(\mathcal{S}) \mid a \in \Sigma\}$-team automaton over $\mathcal{S}$ is called the maximal-syn team automaton *(over $\mathcal{S}$) and*

(2) an action $a \in \Sigma$ is called maximal-syn *in $\mathcal{T}$ if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$.* □

5.4.1 Homogeneous Versus Heterogeneous

The team automata from Definitions 5.4.3 and 5.4.10(1) differ by the type of predicate that needs to be satisfied. However, it is one and the same predicate that needs to be satisfied by *all* external actions. Such team automata are called *homogeneous*, as opposed to team automata for which different subsets of external actions satisfy (potentially) different predicates, which are called *heterogeneous*.

When defining heterogeneous team automata we need to specify exactly which (combinations of) predicates must hold for which subsets of external actions. Consider, e.g., that we want to construct a team automaton over $\mathcal{S}$ such that *all* of its input actions are *ai*, while *all* of its locally-controlled actions are *ms*. Then we construct the $\{\mathcal{R}_a^{ai}(\mathcal{S}) \mid a \in \Sigma_{inp}\} \cup \{\mathcal{R}_a^{ms}(\mathcal{S}) \mid a \in \Sigma_{loc}\}$-team automaton over $\mathcal{S}$, which is thus an example of a heterogeneous team automaton.

Example 5.4.11. (Example 4.2.8 continued) We turn the automata $\mathcal{A}_1$ and $\mathcal{A}_2$, depicted in Figure 4.7(a), into component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, by distributing their respective alphabets over input, output, and internal alphabets. We let a and b be input actions in $\mathcal{C}_1$ and we let a be an output action in $\mathcal{C}_2$. Consequently, $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2\}$ is a composable system. Note that any team automaton over $\mathcal{S}$ will have input alphabet $\{b\}$, output alphabet $\{a\}$, and an empty internal alphabet.

We now construct a homogeneous team automata over $\mathcal{S}$. The $\{\mathcal{R}_c^{sms}(\mathcal{S}) \mid c \in \Sigma\}$-team automaton $\mathcal{T}^1$ (i.e. the *maximal-sms* team automaton) over $\mathcal{S}$ is depicted in Figure 5.12(a).

It is easy to construct other homogeneous team automata over $\mathcal{S}$. The $\{\mathcal{R}_c^{ms}(\mathcal{S}) \mid c \in \Sigma\}$-team automaton over $\mathcal{S}$, e.g., is obtained by adding the transition $((q_1', q_2), a, (q_1', q_2'))$ to the transition relation of $\mathcal{T}^1$. The resulting *maximal-ms* team automaton $\mathcal{T}^2$ is depicted in Figure 5.12(b).

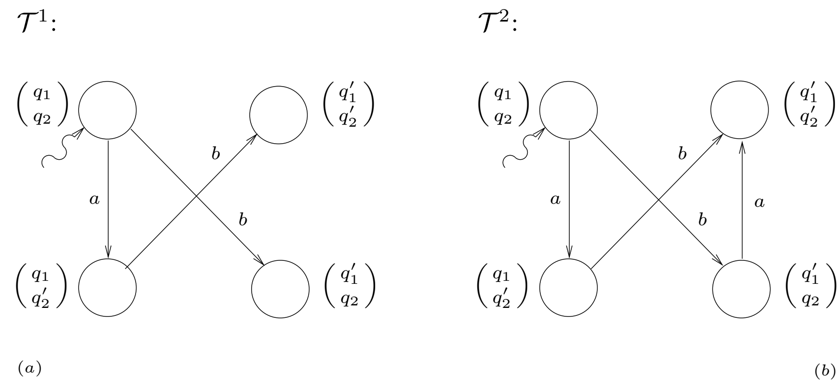


Fig. 5.12. Team automata $\mathcal{T}^1$ and $\mathcal{T}^2$.

It is also not difficult to construct heterogeneous team automata over $\mathcal{S}$. The $\{\mathcal{R}_c^{free}(\mathcal{S}) \mid c \in \Sigma_{inp}\} \cup \{\mathcal{R}_c^{ai}(\mathcal{S}) \mid c \in \Sigma_{out}\} \cup \{\Delta_c(\mathcal{S}) \mid c \in \Sigma_{int}\}$-team automaton over $\mathcal{S}$, e.g., is the team automaton $\mathcal{T}^1$ depicted in Figure 5.12(a). This is thus an example of a team automaton that is both homogeneous and heterogenous. $\square$

As this example has shown, the dividing line between homogeneous and heterogeneous team automata is very thin.

We have paved the way for even more specific team automata that lie inbetween homogeneous and heterogeneous team automata, since we can also construct, e.g., the $\{\mathcal{R}_a^{sopp}(\mathcal{S}) \cap \mathcal{R}_a^{ms}(\mathcal{S}) \mid a \in \Sigma_{ext}\} \cup \{\Delta_a(\mathcal{S}) \mid a \in \Sigma_{int}\}$-team automaton over $\mathcal{S}$ or the $\{\mathcal{R}_a^{ai}(\mathcal{S}) \mid a \in \Sigma_{inp}\} \cup \{\mathcal{R}_a^{sopp}(\mathcal{S}) \cap \mathcal{R}_a^{ms}(\mathcal{S}) \mid a \in \Sigma_{out}\} \cup \{\Delta_a(\mathcal{S}) \mid a \in \Sigma_{int}\}$-team automaton over $\mathcal{S}$.

To conclude this section we make the observation that, given a composable system $\mathcal{S}$, there exist team automata over $\mathcal{S}$ that cannot be obtained as the homogeneous team automaton of any of the types introduced above. Shortly we will give an example of one such a team automaton. We moreover conjecture that it does not help to consider heterogeneous team automata. In other words, there exist team automata over $\mathcal{S}$ whose transition relations cannot be obtained as the result of any combination of the predicates introduced in Definitions 4.5.2, 5.4.4, 5.4.5, and 5.4.7.

Example 5.4.12. (Example 5.4.11 continued) Let $\mathcal{T}^3$ be obtained by removing the transition $((q_1, q_2), b, ((q_1', q_2))$ from the transition relation of $\mathcal{T}^2$. Now $\mathcal{T}^3$ is clearly a team automaton over $\mathcal{S}$. However, it is straightforward to verify that $\mathcal{T}^3$ cannot be obtained as the homogeneous team automaton defined by any of the predicates introduced in Definitions 4.5.2, 5.4.4, 5.4.5, and 5.4.7.

Furthermore, it seems unlikely that — given the current predicates — $\mathcal{T}^3$ can be obtained as a heterogeneous team automaton over $\mathcal{S}$. Intuitively, the reason for this resides in the fact that in $\mathcal{T}^3$, b is its only input action, its output domain is empty, and as far as its input domain is concerned, transitions $((q_1, q_2), b, ((q_1', q_2))$ and $((q_1, q_2'), b, (q_1', q_2'))$ cannot be distinguished. It thus appears to be the case that any team automaton over $\mathcal{S}$ that is constructed according to any (combination) of the predicates introduced in Definitions 4.5.2, 5.4.4, 5.4.5, and 5.4.7 will either contain none of the two b-transitions above, or both. □

Summarizing, in this section we have shown that there exists a large variety of combinations of types of synchronizations that can be used to model many intricate interactions among system components. Given that those components are modeled by component automata and that the interactions the system should exhibit are known, a designer can choose how to construct the unique team automaton over the component automata as a model of the system he or she set out to design.

5.5 Effect of Synchronizations

The (maximal) types of synchronization introduced earlier in this chapter, together with the (maximal) types of synchronization introduced in Sections 4.4 and 4.5, form a whole range of possible synchronizations within team automata. In Section 4.6 we studied the effect that the basic synchronizations *free*, *ai*, *si*, and their maximal variants have on the inheritance of the automata-theoretic properties of Section 3.2 from synchronized automata to their (sub)automata, and vice versa. In this section we extend this study to team automata, i.e. we now take into account that we deal with alphabets with a distinction into three distinct types of actions. We apply some restrictions, though.

First we do not extend this study to incorporate also the more complex types of synchronization introduced earlier on in this chapter. As already mentioned in the Introduction, such a full study is beyond the scope of this thesis. What we do provide is a systematic study of the role *free*, *ai*, and *si* actions play in our approach of modeling collaboration between system components through synchronizations of actions shared by these components.

Secondly, we do not take into account the properties action reducedness, transition reducedness, and state reducedness. Again, such a full study is beyond the scope of this thesis. Instead we focus on the inheritance of enabling and determinism from team automata to their constituents, and vice versa.

To this aim, the results of Section 4.6 are carried over to team automata, after which we study the specific role of the distinction of the set of actions of a team automaton into input, ouput, and internal actions. It turns out that we need to be particularly careful concerning the possibility of an action being input to a component automaton from $\mathcal{S}$ and output to the team automata over $\mathcal{S}$.

We start this section with a study of the top-down inheritance — from team automata to their subteams and component automata — of enabling and determinism. Subsequently we investigate also the bottom-up preservation — from subteams and component automata to team automata.

Notation 8. *For the remainder of this chapter we let $\Sigma_{i,ext}$ denote the set of external actions $\Sigma_{i,inp} \cup \Sigma_{i,out}$ of our fixed component automaton $\mathcal{C}_i$, where $i \in \mathcal{I}$, and we let $\Sigma_{i,loc}$ denote its set of locally-controlled actions $\Sigma_{i,out} \cup \Sigma_{i,int}$. Recall that Σ_i denotes its set of actions $\Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$. Furthermore, we fix an arbitrary $j \in \mathcal{I}$ and an arbitrary subset $J \subseteq \mathcal{I}$. We let $\Sigma_{J,ext}$ denote the set of external actions $\Sigma_{J,inp} \cup \Sigma_{J,out}$ of the subteam SUB_J of $\mathcal{T}$ and we let $\Sigma_{J,loc}$ denote its set of locally-controlled actions $\Sigma_{J,out} \cup \Sigma_{J,int}$. Recall that Σ_J denotes its set of actions $\Sigma_{J,inp} \cup \Sigma_{J,out} \cup \Sigma_{J,int}$. Finally, recall that Σ denotes the set of actions $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$, Σ_{ext} denotes the set of external actions $\Sigma_{inp} \cup \Sigma_{out}$, and Σ_{loc} denotes the set of locally-controlled actions $\Sigma_{out} \cup \Sigma_{int}$ of any team automaton over our fixed composable system $\mathcal{S}$.* □

5.5.1 Top-Down Inheritance of Properties

In this subsection we search for sufficient conditions under which enabling and determinism are inherited from team automata to their subteams and component automata.

It is clear that Definitions 3.2.42 and 3.2.57 extend in a natural way to component automata. Given an alphabet Θ disjoint from the set of states, we can thus speak of a Θ-*enabling component automaton* and of a Θ-*deterministic component automaton*. Moreover, if Θ equals its set of actions, then we simply speak of enabling and deterministic component automata, respectively.

Finally, recall from Theorem 5.4.2 that for all $a \in \Sigma_{int}$, we know that $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $syn \in \{no, free, ai, si\}$.

Enabling

In case the distribution of the alphabet plays no role, then the results concerning the inheritance of enabling from team automata to their subteams and component automata can obviously be lifted from Theorem 4.6.19.

Theorem 5.5.1. *Let $\mathcal{T}$ be Θ-enabling. Then*

(1) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_J$, then SUB_J is Θ-enabling, and

(2) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{C}_j$ is Θ-enabling. □

Since $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$ and $\Sigma_{alph} \cap \Sigma_j \subseteq \Sigma_{j,alph}$, for $alph \in \{inp, int, ext\}$, the following result follows immediately.

Corollary 5.5.2. *Let $alph \in \{inp, int, ext\}$ and let $\mathcal{T}$ be Σ_{alph}-enabling. Then*

(1) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma_{J,alph}$, then SUB_J is Σ_{alph}-enabling, and

(2) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma_{j,alph}$, then $\mathcal{C}_j$ is Σ_{alph}-enabling. □

Note that this corollary does not cover the cases in which $alph \in \{out, loc\}$. In the following example we show that the fact that a team automaton $\mathcal{T}$ over $\mathcal{S}$ is Σ_{out}-enabling in general does not imply that each of its subteams (component automata from $\mathcal{S}$) is Σ_{out}-enabling, not even if all its output actions are *ai* in $\mathcal{T}$.

Example 5.5.3. (Example 4.2.1 continued) We turn automata $\mathcal{A}_2$ and $\mathcal{A}_3$ into component automata $\mathcal{C}_2$ and $\mathcal{C}_3$, respectively, by making a an output action of $\mathcal{C}_2$ and an input action of $\mathcal{C}_3$. The other elements of $\mathcal{C}_2$ and $\mathcal{C}_3$ are as in their underlying automata depicted in Figure 4.6(a). Then $\{\mathcal{C}_2, \mathcal{C}_3\}$ is a composable system and any team automaton $\mathcal{T}$ over $\{\mathcal{C}_2, \mathcal{C}_3\}$ has output alphabet $\{a\}$, while its input as well as its internal alphabet is empty.

Consequently, let $\mathcal{T}$ be the team automaton whose underlying synchronized automaton is depicted in Figure 4.6(b) once states (p, q, r) and (p, q, r') have been replaced by states (q, r) and (q, r'), respectively. Clearly $\mathcal{T}$ is $\{a\}$-enabling. It is however easy to see that $\mathcal{C}_3$ is not, even though all its output actions trivially (since there are none) are *ai* in $\mathcal{T}$. Moreover, the subteam $SUB_{\{3\}}$ of $\mathcal{T}$ is essentially a copy of $\mathcal{C}_3$ and is thus neither $\{a\}$-enabling. □

An additional condition is needed to extend Corollary 5.5.2 to the cases in which $alph \in \{out, loc\}$.

Corollary 5.5.4. *Let $alph \in \{out, loc\}$ and let $\mathcal{T}$ be Σ_{alph}-enabling. Then*

(1) if $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$ and $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma_{J,alph}$, then SUB_J is Σ_{alph}-enabling, and

(2) if $\Sigma_{alph} \cap \Sigma_j \subseteq \Sigma_{j,alph}$ and $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma_{j,alph}$, then $\mathcal{C}_j$ is Σ_{alph}-enabling. □

Determinism

In case the distribution of the alphabet plays no role, then the results concerning the inheritance of determinism from team automata to their subteams and component automata can obviously be lifted from Theorem 4.6.22.

Theorem 5.5.5. *Let $\mathcal{T}$ be Θ-deterministic and let $syn \in \{no, free, ai, si\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_J$, then SUB_J is Θ-deterministic, and

(2) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$ and each a-transition of $\mathcal{C}_j$ is present in $\mathcal{T}$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{C}_j$ is Θ-deterministic. □

Since $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$ and $\Sigma_{alph} \cap \Sigma_j \subseteq \Sigma_{j,alph}$, for $alph \in \{inp, int, ext\}$, the following result follows immediately.

Corollary 5.5.6. *Let $alph \in \{inp, int, ext\}$ and let $\mathcal{T}$ be Σ_{alph}-deterministic. Let $syn \in \{no, free, ai, si\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Sigma_{J,alph}$, then SUB_J is Σ_{alph}-deterministic, and

(2) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$ and each a-transition of $\mathcal{C}_j$ is present in $\mathcal{T}$, for all $a \in \Sigma_{j,alph}$, then $\mathcal{C}_j$ is Σ_{alph}-deterministic. □

Note that this corollary does not cover the cases in which $alph \in \{out, loc\}$. In the following example we show that the fact that a team automaton $\mathcal{T}$ over $\mathcal{S}$ is Σ_{out}-deterministic in general does not imply that each of its constituting component automata is Σ_{out}-deterministic, not even if all its output actions are *maximal-free*, *maximal-ai*, or *maximal-si* in $\mathcal{T}$ and all component automaton transitions of output actions are present in $\mathcal{T}$. It is not difficult to provide a similar example for the case of subteams.

Example 5.5.7. (Example 4.6.5 continued) We turn automata $\mathcal{A}_1$ and $\mathcal{A}_2$ into component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, by making a an output action of $\mathcal{C}_1$ and an input action of $\mathcal{C}_2$. The other elements of $\mathcal{C}_1$ and $\mathcal{C}_2$ are as in their underlying automata depicted in Figure 4.11. Then $\{\mathcal{C}_1, \mathcal{C}_2\}$ is a composable system and any team automaton $\mathcal{T}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ has output alphabet $\{a\}$, while its input as well as its internal alphabet is empty.

Now let $\mathcal{T}$ be the team automaton with empty transition relation. Hence $\mathcal{T}$ is trivially $\{a\}$-deterministic. It is however clear that $\mathcal{C}_2$ is not, even though all its output actions trivially (since there are none) are *maximal-free*, *maximal-ai*, and *maximal-si* in $\mathcal{T}$ and all its transitions of output actions trivially (again, there are none) are present in $\mathcal{T}$. □

An additional condition is needed to extend Corollary 5.5.6 to the cases in which $alph \in \{out, loc\}$.

Corollary 5.5.8. *Let alph* $\in \{out, loc\}$ *and let* $\mathcal{T}$ *be* Σ_{alph}*-deterministic. Let syn* $\in \{no, free, ai, si\}$*. Then*

(1) if $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$ *and* $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$*, for all* $a \in \Sigma_{J,alph}$*, then* SUB_J *is* Σ_{alph}*-deterministic, and*

(2) if $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$*,* $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$*, and each a-transition of* $\mathcal{C}_j$ *is present in* $\mathcal{T}$*, for all* $a \in \Sigma_{j,alph}$*, then* $\mathcal{C}_j$ *is* Σ_{alph}*-deterministic.* □

5.5.2 Bottom-Up Inheritance of Properties

Dual to the above investigations we now change focus and study the sufficient conditions under which enabling and determinism are preserved from component automata from $\mathcal{S}$ to team automata over $\mathcal{S}$.

We recall from Section 5.2 that $\mathcal{T}$ is a team automaton over $\mathcal{S}'$ — upto a reordering — whenever $\mathcal{S}' = \{SUB_{\mathcal{I}_j} \mid \{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}\}$. Hence it suffices to investigate the conditions under which the enabling and determinism of (component automata from) a composable system is preserved by a team automaton over that composable system.

Enabling

In case the distribution of the alphabet plays no role, then the results concerning the preservation of enabling from component automata from $\mathcal{S}$ to a team automaton over $\mathcal{S}$ can obviously be lifted from Theorem 4.6.33.

Theorem 5.5.9. *Let* $\mathcal{C}_j$ *be* Θ*-enabling. Then*

if each a-transition of $\mathcal{C}_j$*, for all* $a \in \Theta$*, is omnipresent in* $\mathcal{T}$*, then* $\mathcal{T}$ *is* $\Theta \cap \Sigma_j$*-enabling.* □

As the set of input (output, internal) actions of any team automaton $\mathcal{T}$ over $\mathcal{S}$ is included in the union of the sets of input (output, internal) actions of the component automata from $\mathcal{S}$, we immediately obtain the following result.

Corollary 5.5.10. *Let alph* $\in \{inp, out, int, ext, loc\}$ *and let* $\mathcal{C}_i$ *be* $\Sigma_{i,alph}$*-enabling, for all* $i \in \mathcal{I}$*. Then*

if all a-transitions of $\mathcal{C}_i$*, for all* $a \in \Sigma_{i,alph}$ *and for all* $i \in \mathcal{I}$*, are omnipresent in* $\mathcal{T}$*, then* $\mathcal{T}$ *is* Σ_{alph}*-enabling.* □

Note how, contrary to the results in the previous subsection, the possibility of an action being input to a component automaton from $\mathcal{S}$ and output to the team automata over $\mathcal{S}$ plays no role here. The reason is the fact that every input (output) action of a team automaton $\mathcal{T}$ over $\mathcal{S}$ needs to be an input (output) action of at least one component automaton from $\mathcal{S}$. Hence no additional condition is needed to cover the case in which $alph \in \{out, loc\}$ in this corollary. Even though an input action a of a non-$\{a\}$-enabling component automaton from $\mathcal{S}$ may be an output action of $\mathcal{T}$, it cannot prevent $\mathcal{T}$ from being Σ_{out}-enabling if the conditions of this corollary are satisfied. The reason is that according to these conditions, the component automaton from $\mathcal{S}$ in which a appears as an output action must not only be $\{a\}$-enabling, but all its a-transitions must moreover be omnipresent in $\mathcal{T}$.

Determinism

In case the distribution of the alphabet plays no role, then the results concerning the preservation of determinism from component automata from $\mathcal{S}$ to a team automaton over $\mathcal{S}$ can obviously be lifted from Theorem 4.6.35.

Theorem 5.5.11. *Let $\mathcal{S}$ be Θ-deterministic and let $syn \in \{ai, si\}$. Then*

if $\delta_a \subseteq \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma$, then $\mathcal{T}$ is Θ-deterministic. □

Since $\Sigma_{alph} \cap \Sigma_j \subseteq \Sigma_{j,alph}$, for $alph \in \{inp, int, ext\}$, the following result follows immediately.

Lemma 5.5.12. *Let $alph \in \{inp, int, ext\}$. Then*

if $\mathcal{C}_j$ is $\Sigma_{j,alph}$-deterministic, then $\mathcal{C}_j$ is Σ_{alph}-deterministic. □

Since an action may be input in a component automaton from $\mathcal{S}$ but output in a team automaton over $\mathcal{S}$, Lemma 5.5.12 cannot be extended to the cases in which $alph \in \{out, loc\}$. To see this, consider an external action a that is input to a component automaton (e.g. $\mathcal{C}_2$) which is $\Sigma_{2,out}$-deterministic but not $\{a\}$-deterministic, and output to another component automaton (e.g. $\mathcal{C}_1$). Then a will clearly be an output action of any team automaton over $\{\mathcal{C}_1, \mathcal{C}_2\}$, but $\mathcal{C}_2$ nevertheless is not $\{a\}$-deterministic.

Lemma 5.5.12 allows us to extend Theorem 5.5.11 to the cases in which $alph \in \{inp, int, ext\}$.

Corollary 5.5.13. *Let $alph \in \{inp, int, ext\}$ and let $\mathcal{C}_i$ be $\Sigma_{i,alph}$-deterministic, for all $i \in \mathcal{I}$. Let $syn \in \{ai, si\}$. Then*

if $\delta_a \subseteq \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Sigma_{alph}$, then $\mathcal{T}$ is Σ_{alph}-deterministic. □

Note that — contrary to Corollary 5.5.10 — Corollary 5.5.13 cannot be extended to the cases in which *alph* $\in \{out, loc\}$, not even when we consider team automata whose every action is *maximal-free*, *maximal-ai*, or *maximal-si*. This is because the transitions that cause a component automaton not to be deterministic are not a priori excluded from being present in such team automata, but when they are present they thus also cause those team automata not to be deterministic. In the following example we demonstrate this by showing that even if $\mathcal{C}_i$ is $\Sigma_{i,out}$-deterministic, for all $i \in \mathcal{I}$, and $\delta_a \subseteq \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Sigma_{out}$ and $syn \in \{free, ai, si\}$, then this in general does not imply that $\mathcal{T}$ is Σ_{out}-deterministic.

Example 5.5.14. Let component automata $\mathcal{C}_1 = (\{q_1, q_1'\}, (\{a\}, \varnothing, \varnothing), \{(q_1, a, q_1), (q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{C}_2 = (\{q_2, q_2'\}, (\varnothing, \{a\}, \varnothing), \{(q_2, a, q_2')\}, \{q_2\})$ be as depicted in Figure 5.13.

Fig. 5.13. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$.

Note that both $\mathcal{C}_i$, with $i \in [2]$, are $\Sigma_{i,out}$-deterministic. Furthermore, $\{\mathcal{C}_i \mid i \in [2]\}$ is a composable system. Now consider the team automaton $\mathcal{T} = (Q, (\varnothing, \{a\}, \varnothing), \delta, \{(q_1, q_2)\})$, where $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$ and $\delta = \{((q_1, q_2), a, (q_1, q_2')), ((q_1, q_2), a, (q_1', q_2'))\}$, over this composable system. It is depicted in Figure 5.14(a).

Clearly $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \subseteq \mathcal{R}_a^{si}(\mathcal{S})$, but $\mathcal{T}$ obviously is not Σ_{out}-deterministic.

Next consider the team automaton $\mathcal{T}' = (Q, (\varnothing, \{a\}, \varnothing), \delta', \{(q_1, q_2)\})$, where $\delta' = \{((q_1, q_2), a, (q_1, q_2)), ((q_1, q_2), a, (q_1', q_2))\}$, over this composable system. It is depicted in Figure 5.14(b). Clearly $\delta_a' \subseteq \mathcal{R}_a^{free}(\mathcal{S})$. However, also $\mathcal{T}'$ obviously is not Σ_{out}-deterministic. □

5.6 Inheritance of Synchronizations

In this section we start an initial exploration into the conditions under which the types of synchronization introduced in Sections 5.3 and 5.4 are inherited top-down — from team automata to subteams — and preserved bottom-up —

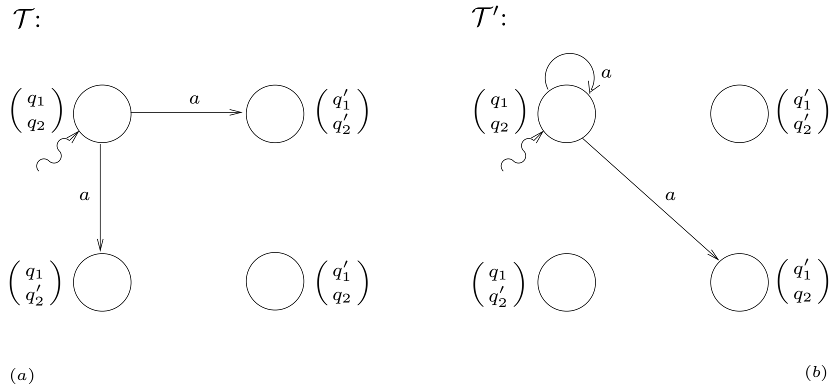


Fig. 5.14. Team automata $\mathcal{T}$ and $\mathcal{T}'$.

from subteams to team automata — as an addition to the results presented in Section 4.7 on the inheritance and preservation of the basic synchronizations *free*, *ai*, and *si*.

Since we deal with synchronizations *between* component automata constituting a team automaton, there is no need to study whether synchronizations are inherited by component automata from team automata — and vice versa: in any component automaton — and in any team automaton over a single component automaton — all its input (output) actions trivially are *sipp* and *wipp* (*sopp* and *wopp*) while all its output actions trivially are *ms*, *sms*, and *wms*.

We begin by considering the inheritance of the peer-to-peer types of synchronization. In the following example we show that if an action is *sipp* (*wipp*, *sopp*, *wopp*) in a team automaton, then this in general does not imply that it is also *sipp* (*wipp*, *sopp*, *wopp*) in each of its subteams.

Example 5.6.1. Consider the composable system $\{\mathcal{C}_1, \mathcal{C}_2\}$, which consists of component automata $\mathcal{C}_1 = (\{q_1, q_1'\}, (\{a\}, \varnothing, \varnothing), \{(q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{C}_2 = (\{q_2, q_2'\}, (\varnothing, \{a\}, \varnothing), \{(q_2, a, q_2')\}, \{q_2\})$. It is depicted in Figure 5.15(a).

Now consider team automaton $\mathcal{T} = (\{(q_1, q_2), (q_1', q_2), (q_1, q_2'), (q_1', q_2')\}, (\varnothing, \{a\}, \varnothing), \{((q_1, q_2), a, (q_1', q_2'))\}, \{(q_1, q_2)\})$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$, depicted in Figure 5.15(b).

Clearly $\mathcal{I}_{a,inp}(\{\mathcal{C}_1, \mathcal{C}_2\}) = \{1\}$ and $\mathcal{I}_{a,out}(\{\mathcal{C}_1, \mathcal{C}_2\}) = \{2\}$. Thus a trivially is *ai* in both $SUB_{a,inp} = SUB_{\{1\}}$ and $SUB_{a,out} = SUB_{\{2\}}$. Hence a is *sipp* and *sopp* (and thus *wipp* and *wopp*) in $\mathcal{T}$.

We observe that $SUB_{\{2\}_{a,inp}(\{\mathcal{C}_2\})}(SUB_{\{2\}}) = (\varnothing, (\varnothing, \varnothing, \varnothing), \varnothing, \varnothing) = SUB_{\{1\}_{a,out}(\{\mathcal{C}_1\})}(SUB_{\{1\}})$. This implies $a \notin SI(SUB_{\{2\}_{a,inp}(\{\mathcal{C}_2\})}(SUB_{\{2\}}))$

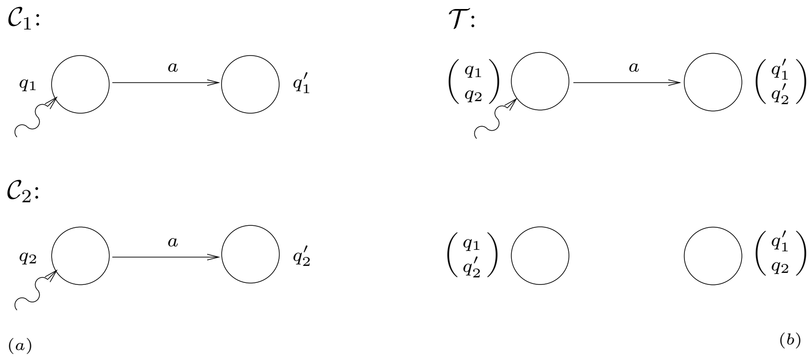


Fig. 5.15. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, and team automaton $\mathcal{T}$.

and $a \notin SI(SUB_{\{1\}_{a,out}(\{\mathcal{C}_1\})}(SUB_{\{1\}}))$. Hence a is neither *wipp* nor *wopp* (and thus neither *sipp* nor *sopp*) in $SUB_{\{2\}}$ and $SUB_{\{1\}}$, respectively. $\square$

From Lemma 4.7.1(2,3) we obtain that *sipp* and *wipp* (*sopp* and *wopp*) actions *are* inherited from a team automaton to a subteam as long as the subteam is chosen from the input (output) domain of the team automaton. Recall that $\Sigma_{out} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$.

Lemma 5.6.2. *Let* $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$ *and let* $\varnothing \neq K \subseteq \mathcal{I}_{a,inp}(\mathcal{S})$. *Then*

(1) if $a \in SIPP(\mathcal{T})$, *then* $a \in SIPP(SUB_K(\mathcal{T}))$, *and*

(2) if $a \in WIPP(\mathcal{T})$, *then* $a \in WIPP(SUB_K(\mathcal{T}))$.

Let $a \in \Sigma_{out}$ *and let* $\varnothing \neq L \subseteq \mathcal{I}_{a,out}(\mathcal{S})$. *Then*

(3) if $a \in SOPP(\mathcal{T})$, *then* $a \in SOPP(SUB_L(\mathcal{T}))$, *and*

(4) if $a \in WOPP(\mathcal{T})$, *then* $a \in WOPP(SUB_L(\mathcal{T}))$.

Proof. (1) From $a \in \Sigma_{inp}$ and $\varnothing \neq K \subseteq \mathcal{I}_{a,inp}(\mathcal{S})$ we know that the input domain of a in $\{\mathcal{C}_k \mid k \in K\}$ is K itself. Hence $K_{a,inp}(\{\mathcal{C}_k \mid k \in K\}) = K \neq \varnothing$. Now let a be *sipp* in $\mathcal{T}$. Then by Definition 5.3.4(1), a is *ai* in $SUB_{\mathcal{I}_{a,inp}(\mathcal{S})}(\mathcal{T})$. Since $K \subseteq \mathcal{I}_{a,inp}(\mathcal{S})$, Lemma 4.7.1(2) directly implies that a is *ai* in $SUB_K(SUB_{\mathcal{I}_{a,inp}(\mathcal{S})}(\mathcal{T})) = SUB_{K_{a,inp}(\{\mathcal{C}_k \mid k \in K\})}(SUB_K(\mathcal{T}))$. Definition 5.3.4(1) now implies that a is *sipp* in $SUB_K(\mathcal{T})$.

(2-4) Analogous. $\square$

Next we wonder whether *sipp* and *wipp* (*sopp* and *wopp*) actions are preserved from subteams to team automata. In the following example we show that in general they are not.

Example 5.6.3. (Example 5.3.18 continued) Note that $a \notin WOPP(\mathcal{T}) \cup SOPP(\mathcal{T})$ since $a \notin SI(SUB_{a,out})$. However, it is easy to see that $a \in SOPP(SUB_{\{1\}}(\mathcal{T})) \subseteq WOPP(SUB_{\{1\}}(\mathcal{T}))$. It is not difficult to adjust this example in order to show that also *sipp* and *wipp* actions in general are not preserved from subteams to team automata. □

It turns out that *sipp* and *wipp* (*sopp* and *wopp*) actions *are* preserved from the input (output) subteam of a team automaton to the team automaton as a whole, which together with the previous lemma provides us with the following result.

Theorem 5.6.4. *Let* $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$*, let* $K = \mathcal{I}_{a,inp}(\mathcal{S})$*, and let* $SUB_K(\mathcal{T}) = (Q_K, \Sigma_K, \delta_K, I_K)$*. Then*

(1) $a \in \Sigma_K \cap SIPP(\mathcal{T})$ *if and only if* $a \in SIPP(SUB_K(\mathcal{T}))$ *and*

(2) $a \in \Sigma_K \cap WIPP(\mathcal{T})$ *if and only if* $a \in WIPP(SUB_K(\mathcal{T}))$*.*

Let $a \in \Sigma_{out}$*, let* $L = \mathcal{I}_{a,out}(\mathcal{S})$*, and let* $SUB_L(\mathcal{T}) = (Q_L, \Sigma_L, \delta_L, I_L)$*. Then*

(3) $a \in \Sigma_L \cap SOPP(\mathcal{T})$ *if and only if* $a \in SOPP(SUB_L(\mathcal{T}))$ *and*

(4) $a \in \Sigma_L \cap WOPP(\mathcal{T})$ *if and only if* $a \in WOPP(SUB_L(\mathcal{T}))$*.*

Proof. (1) (Only if) Directly from Lemma 5.6.2(1).

(If) Let $a \in SIPP(SUB_K(\mathcal{T}))$. Then Definition 5.3.4(1) implies that $a \in \Sigma_K \cap AI(SUB_K(\mathcal{T})))$. Since $K = \mathcal{I}_{a,inp}(\mathcal{S})$ and $a \in \Sigma_K \cap AI(SUB_K(\mathcal{T}))$, Definition 5.3.4(1) implies that $a \in \Sigma_K \cap SIPP(\mathcal{T})$.

(2-4) Analogous. □

Finally, we turn to the master-slave types of synchronization. In the following example we show that if an action is *ms* (*sms*, *wms*) in a team automaton, then this in general does not imply that it is also *ms* (*sms*, *wms*) in each of its subteams.

Example 5.6.5. (Example 5.6.1 continued) Clearly a is *sms* (and thus also *ms* and *wms*) in $\mathcal{T}$. However, a is not an output action of $SUB_{\{1\}}$ and it thus cannot be *ms* (and hence neither *sms* nor *wms*) in $SUB_{\{1\}}$. □

We do have that every output action a of $\mathcal{T}$ is *ms* in any subteam of $\mathcal{T}$ determined by a subset of the output domain of a in $\mathcal{S}$.

Theorem 5.6.6. *If* $a \in \Sigma_{out}$ *and* $\varnothing \neq K \subseteq \mathcal{I}_{a,out}(\mathcal{S})$*, then* $a \in MS(SUB_K(\mathcal{T}))$*.*

Proof. Let $a \in \Sigma_{out}$ and let $\varnothing \neq K \subseteq \mathcal{I}_{a,out}(\mathcal{S})$. Clearly $a \in \Sigma_{K,out}$. In fact, the output domain $J = K_{a,out}(\{\mathcal{C}_k \mid k \in K\})$ of a in $\{\mathcal{C}_k \mid k \in K\}$ is K itself. Now let $(p,p') \in (\delta_K)_a = \mathrm{proj}_K{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_k \mid k \in K\})$. Then $\mathrm{proj}_K{}^{[2]}(p,p') = (p,p')$ and it thus follows from the above that $\mathrm{proj}_J{}^{[2]}((\delta_K)_a) = (\delta_K)_a = (\delta_J)_a$. Hence $a \in MS(SUB_K(\mathcal{T}))$. □

We also get that an *ms* action a from a team automaton over $\mathcal{S}$ is also *ms* in all subteams determined by a set that contains the output domain of a in $\mathcal{S}$.

Theorem 5.6.7. *If* $a \in MS(\mathcal{T})$ *and* $K \supseteq \mathcal{I}_{a,out}(\mathcal{S})$, *then* $a \in MS(SUB_K(\mathcal{T}))$.

Proof. Let $a \in MS(\mathcal{T})$ and let $K \supseteq \mathcal{I}_{a,out}(\mathcal{S})$. Clearly $a \in \Sigma_{out}$ and hence $\mathcal{I}_{a,out}(\mathcal{S}) \neq \varnothing$. Now let $(p,p') \in (\delta_K)_a$. Then there must exist $q, q' \in Q$ such that $(q,q') \in \delta_a$ and $\mathrm{proj}_K{}^{[2]}(q,q') = (p,p') \in \Delta_a(\{\mathcal{C}_k \mid k \in K\})$. Since $a \in MS(\mathcal{T})$, there exists a $k \in \mathcal{I}_{a,out}(\mathcal{S}) \subseteq K$ such that $\mathrm{proj}_k{}^{[2]}(q,q') = \mathrm{proj}_k{}^{[2]}(p,p') \in \delta_{k,a}$. Because the output domain $J = K_{a,out}(\{\mathcal{C}_k \mid k \in K\})$ of a in $\{\mathcal{C}_k \mid k \in K\}$ is $\mathcal{I}_{a,out}(\mathcal{S})$ it follows that $\mathrm{proj}_J{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_\ell \mid \ell \in J\})$ and thus $\mathrm{proj}_J{}^{[2]}((\delta_K)_a) = (\delta_J)_a$. Hence $a \in MS(SUB_K(\mathcal{T}))$. □

Furthermore, as we show next, an *ms* action a is preserved from a subteam to the team automaton over $\mathcal{S}$ as a whole, provided that the subteam is determined by a set that contains the input domain of a in $\mathcal{S}$.

Theorem 5.6.8. *Let* $a \in \Sigma_{out}$ *and let* $K \supseteq \mathcal{I}_{a,inp}(\mathcal{S})$. *Then*

if $a \in MS(SUB_K(\mathcal{T}))$, *then* $a \in MS(\mathcal{T})$.

Proof. Let $J = \mathcal{I}_{a,out}(\mathcal{S})$. Note that $J \neq \varnothing$. Now let $(q,q') \in \delta_a$ and assume that $\mathrm{proj}_J{}^{[2]}(q,q') \notin (\delta_J)_a$, which means that $\mathrm{proj}_\ell{}^{[2]}(q,q') \notin \delta_{\ell,a}$, for all $\ell \in J$, i.e. only the input domain of a in $\mathcal{S}$ is involved in this transition. Consequently, $\mathrm{proj}_K{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_k \mid k \in K\})$. Now suppose that $a \in MS(SUB_K(\mathcal{T}))$. Then $a \in \Sigma_{K,out}$ and thus $K \cap J \neq \varnothing$. Moreover, from a being *ms* in $SUB_K(\mathcal{T})$ it follows that there exists a $k \in K \cap J$ such that $\mathrm{proj}_k{}^{[2]}(q,q') \in \delta_{k,a}$, a contradiction with the fact that $\mathrm{proj}_J{}^{[2]}(q,q') \notin (\delta_J)_a$. Hence we have proven that $a \notin MS(\mathcal{T})$ implies $a \notin MS(SUB_K(\mathcal{T}))$. □

Finally, we note that whenever an output action a is *sms* (*wms*) in $\mathcal{T}$ and $J \subseteq \mathcal{I}_{a,out}(\mathcal{S})$, then a trivially is *sms* (*wms*) in $SUB_J(\mathcal{T})$ because the input domain of a in $\{\mathcal{C}_j \mid j \in J\}$ is empty.

This completes our initial exploration into the conditions under which the complex types of synchronization introduced in Section 5.3 are inherited from team automata to subteams, and vice versa.

We conclude this section with a result on the inheritance of the maximal types of synchronization introduced in Section 5.4. Using our knowledge from earlier results of this section we extend the results presented in Theorem 4.7.5 to the case of peer-to-peer and master-slave types of synchronization.

Theorem 5.6.9. *Let* $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$ *and let* $K \subseteq \mathcal{I}_{a,inp}(\mathcal{S})$*. Then*

(1) if $\delta_a = \mathcal{R}_a^{sipp}(\mathcal{S})$*, then* $(\delta_K)_a = \mathcal{R}_a^{sipp}(\{\mathcal{C}_k \mid k \in K\})$*, and*

(2) if $\delta_a = \mathcal{R}_a^{wipp}(\mathcal{S})$*, then* $(\delta_K)_a = \mathcal{R}_a^{wipp}(\{\mathcal{C}_k \mid k \in K\})$*.*

Let $a \in \Sigma_{out}$ *and let* $L \subseteq \mathcal{I}_{a,out}(\mathcal{S})$*. Then*

(3) if $\delta_a = \mathcal{R}_a^{sopp}(\mathcal{S})$*, then* $(\delta_L)_a = \mathcal{R}_a^{sopp}(\{\mathcal{C}_\ell \mid \ell \in L\})$*,*

(4) if $\delta_a = \mathcal{R}_a^{wopp}(\mathcal{S})$*, then* $(\delta_L)_a = \mathcal{R}_a^{wopp}(\{\mathcal{C}_\ell \mid \ell \in L\})$*, and*

(5) if $\delta_a = \mathcal{R}_a^{ms}(\mathcal{S})$*, then* $(\delta_L)_a = \mathcal{R}_a^{ms}(\{\mathcal{C}_\ell \mid \ell \in L\})$*.*

Proof. (1) By Lemma 5.6.2(1) we only need to prove that $\delta_a = \mathcal{R}_a^{sipp}(\mathcal{S})$ implies $\mathcal{R}_a^{sipp}(\{\mathcal{C}_k \mid k \in K\}) \subseteq (\delta_K)_a$. Hence let $(p, p') \in \mathcal{R}_a^{sipp}(\{\mathcal{C}_k \mid k \in K\})$. Then by Definition 5.4.4(1) there exists a $(q, q') \in \mathcal{R}_a^{sipp}(\mathcal{S})$ such that $\mathrm{proj}_K{}^{[2]}(q, q') = (p, p')$ and thus, since $\delta_a = \mathcal{R}_a^{sipp}(\mathcal{S})$, $(p, p') = \mathrm{proj}_K{}^{[2]}(q, q') \in (\delta_K)_a$.

(2-4) Analogous.

(5) Analogous, but now using Theorem 5.6.6 and Definition 5.4.7(1). □

5.7 Conclusion

Team automata can be classified on basis of the properties of their transition relations or by imposing conditions on their transition relations, which may lead to team automata that are maximal with respect to the given conditions. Furthermore, we can consider properties at the team level, or at the level of subteams.

Team automata allow exact descriptions of certain groupware notions which may otherwise have an ambiguous interpretation. Consider, e.g., the distinction between cooperation and collaboration within the team automaton model as described in [Ell97]:

“A Team Automaton is defined to be *cooperating* if it is structured so that one of its components is the active master, and all the others are passive slaves.”

and

“A Team Automaton is defined to be *collaborating* if it is structured so that all of the automata are active peers.”

To this it is added that the master-slave mechanism is referred to as *passive cooperation*, since the master is never blocked waiting for a slave. This contrasts with the peer-to-peer mechanism, in which blocking may occur when not all of the participants are ready to execute the action, and which is called *active collaboration*.

The framework of team automata clearly allows for more and finer distinctions. This is mainly due to the uniform approach towards the formalization of the notion of obligation for component automata to participate in the execution of a certain action, which is independent of the role of that action (input or output, peer, master, or slave).

We have thus provided two global interpretations of collaboration through the notions of *ai* (comparable to the adjective "active" above, as blocking may occur) and *si*. Here the input role an action may have is not yet separated from its output role. When this distinction is made we arrive at the four notions of strong (weak) input (output) peer-to-peer.

Cooperation, on the other hand, is formalized through the notions of (weak and strong) *ms* synchronizations. When an action is *ms*, then it cannot be executed as an input action without being simultaneously executed as an output action. In the strong case, all slaves (the component automata having the action as an input action) should participate in the action, whereas in the weak case all component automata that are ready for that action should participate in the synchronization (which corresponds to the "passive" cooperation mentioned above). Note that the master in an *ms* synchronization may be a subteam rather than a single component automaton. As argued in Section 5.2, there is no essential difference between a subteam of a team automaton and a component automaton which itself may have been obtained as a team automaton. Similarly, the slaves may be one or more component automata or one or more subteams.

The above viewpoint also easily allows combinations of cooperation and collaboration, called *hybrids* in [Ell97]. One may, e.g., have an *ms* synchronization in which within the master (subteam) the synchronizations are *sopp*, while the subteam of the slaves exhibits *wipp* synchronization (all slaves that can, participate) or *sipp* synchronization (all slaves have to take part).

Finally, observe that these considerations on cooperation and collaboration all relate to the synchronizations of a single external action. These notions can also be lifted to the level of the team automaton as a whole, either in a homogeneous way or in an heterogeneous way. In the first case there is one type of cooperation or collaboration (the same for all actions) including the identity of the master, the slaves, the input domain, the output domain, etc. In the second case, each external action can have its own cooperation or collaboration specification.

Given requirements for each external action, one may follow the approach outlined in Section 5.4 to construct a unique team automaton with the appropriate combinations of cooperating and collaborating synchronizations.

The theory presented so far has thus led to a flexible framework that allows one to precisely classify, describe and construct many different incarnations of cooperation and collaboration. Which of these may be of use in applications, is for practice to decide.

6. Behavior of Team Automata

In this chapter we study the behavior of team automata. We begin with a few elementary observations on the computational power for the case of finite component automata, i.e. component automata with a finite set of states and a finite alphabet (of input, output, and internal actions). For the rest of this chapter we then turn to component automata and team automata with possibly infinite sets of states and actions. We study the relation between the computations and behavior of team automata on the one hand, and those of their constituting component automata on the other hand. Since a composable system does not uniquely define a team automaton, the relation between the computations and behavior of a team automaton and those of its constituting component automata depends on the allowed synchronizations.

We are particularly interested in conditions which guarantee that a team automaton satisfies *compositionality*. This means that the behavior of a team automaton can be described as a function of the behavior of its constituting component automata. Since component automata and team automata have languages as behavior, we use language-theoretic operations — so called *shuffles* — to describe the combination of words into new words. In order to be able to apply these shuffles in the context of team automata, we extensively investigate their properties in two separate sections. This eventually enables us to identify several types of team automata satisfying compositionality.

6.1 Behavior of Finite Component Automata

Most types of automata considered in this thesis may have an infinite set of states and an infinite set of actions. As already discussed in Section 3.1, by allowing the automata in our framework to have an infinite set of states we end up with automata that have Turing machine power. In this section we study the behavior of *finite component automata*, i.e. of component automata with a finite set of states and a finite alphabet, and — subsequently — the influence that the distinction between input, output, and internal actions has on their behavior (cf. Section 7.1.4).

In the remainder of this section, all component automata have a finite set of states and a finite alphabet. Moreover, we restrict our study to an investigation of their finite computations, and the resulting finitary behavior.

Component automata differ from automata only by the distinction of their set of actions into input, output, and internal actions. In fact, by ignoring this distinction, every finite component automaton $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ can be viewed as an automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$ such that $\Sigma = \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$, with $\mathbf{B}^{\Sigma}_{\mathcal{A}} = \mathbf{B}^{\Sigma}_{\mathcal{C}}$. Conversely, every automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$ such that Q and Σ are finite can be viewed — once its alphabet is disjointly distributed over input, output, and internal actions Σ_1, Σ_2, and Σ_3 — as a component automaton $\mathcal{C} = (Q, (\Sigma_1, \Sigma_2, \Sigma_3), \delta, I)$ such that $\Sigma_1 \cup \Sigma_2 \cup \Sigma_3 = \Sigma$, with $\mathbf{B}^{\Sigma}_{\mathcal{C}} = \mathbf{B}^{\Sigma}_{\mathcal{A}}$.

The computational power of automata with a finite set of states and a finite set of actions equals that of the family of prefix-closed regular finitary languages, which we denote by pREG. The family of regular languages, denoted by REG, is precisely the family of languages accepted by the well-known model of finite (state) automata (cf. the introduction to Chapter 3). Formally, $\mathsf{pREG} = \{L \in \mathsf{REG} \mid L \text{ is prefix closed}\}$. It is known that $\mathsf{pREG} \subset \mathsf{REG}$ and $\mathsf{FIN} \subset \mathsf{REG}$, where FIN denotes the family of finite languages, while FIN and pREG are incomparable.

We denote $\mathsf{CA} = \{\mathbf{B}^{\Sigma}_{\mathcal{C}} \mid \Sigma \text{ is an alphabet and } \mathcal{C} \text{ is a finite component automaton with alphabet } \Sigma\}$. Then the above observations immediately yield the following result.

Lemma 6.1.1. $\mathsf{pREG} = \mathsf{CA}$. □

Note that the inclusion $\mathsf{pREG} \subseteq \mathsf{CA}$ can be proven by choosing any distribution of an automaton's alphabet over input, output, and internal alphabets.

Using this observation once more we now prove that all behavior collected in CA (and hence in pREG) can also be obtained as the input, output, internal, external, and locally-controlled behavior of component automata.

First we introduce some notation. Consider an arbitrary component automaton $\mathcal{C} = (Q, (\Sigma_1, \Sigma_2, \Sigma_3), \delta, I)$ and let $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$. Consequently we set $\mathbf{B}^{inp}_{\mathcal{C}} = \mathbf{B}^{\Sigma_1}_{\mathcal{C}}$, thus $\mathbf{B}^{inp}_{\mathcal{C}} = \mathrm{pres}_{\Sigma_1}(\mathbf{B}^{\Sigma}_{\mathcal{C}})$; $\mathbf{B}^{out}_{\mathcal{C}} = \mathbf{B}^{\Sigma_2}_{\mathcal{C}}$, thus $\mathbf{B}^{out}_{\mathcal{C}} = \mathrm{pres}_{\Sigma_2}(\mathbf{B}^{\Sigma}_{\mathcal{C}})$; $\mathbf{B}^{int}_{\mathcal{C}} = \mathbf{B}^{\Sigma_3}_{\mathcal{C}}$, thus $\mathbf{B}^{int}_{\mathcal{C}} = \mathrm{pres}_{\Sigma_3}(\mathbf{B}^{\Sigma}_{\mathcal{C}})$; $\mathbf{B}^{ext}_{\mathcal{C}} = \mathbf{B}^{\Sigma_1 \cup \Sigma_2}_{\mathcal{C}}$, thus $\mathbf{B}^{ext}_{\mathcal{C}} = \mathrm{pres}_{\Sigma_1 \cup \Sigma_2}(\mathbf{B}^{\Sigma}_{\mathcal{C}})$; $\mathbf{B}^{loc}_{\mathcal{C}} = \mathbf{B}^{\Sigma_2 \cup \Sigma_3}_{\mathcal{C}}$, thus $\mathbf{B}^{loc}_{\mathcal{C}} = \mathrm{pres}_{\Sigma_2 \cup \Sigma_3}(\mathbf{B}^{\Sigma}_{\mathcal{C}})$.

Next we consider the following component automata as variants of $\mathcal{C}$: $[\mathcal{C}, inp] = (Q, (\Sigma, \varnothing, \varnothing), \delta, I)$, $[\mathcal{C}, out] = (Q, (\varnothing, \Sigma, \varnothing), \delta, I)$, and $[\mathcal{C}, int] = (Q, (\varnothing, \varnothing, \Sigma), \delta, I)$.

Lemma 6.1.2. *Let $[\mathcal{C}, inp]$, $[\mathcal{C}, out]$, and $[\mathcal{C}, int]$ be as described above. Then*

(1) $\mathbf{B}^{\Sigma}_{\mathcal{C}} = \mathbf{B}^{inp}_{[\mathcal{C}, inp]} = \mathbf{B}^{out}_{[\mathcal{C}, out]} = \mathbf{B}^{int}_{[\mathcal{C}, int]}$,

(2) $\mathbf{B}^{\Sigma}_{[\mathcal{C},inp]} = \mathbf{B}^{inp}_{[\mathcal{C},inp]} = \mathbf{B}^{ext}_{[\mathcal{C},inp]}$,

(3) $\mathbf{B}^{\Sigma}_{[\mathcal{C},out]} = \mathbf{B}^{out}_{[\mathcal{C},out]} = \mathbf{B}^{ext}_{[\mathcal{C},out]} = \mathbf{B}^{loc}_{[\mathcal{C},out]}$, *and*

(4) $\mathbf{B}^{\Sigma}_{[\mathcal{C},int]} = \mathbf{B}^{int}_{[\mathcal{C},int]} = \mathbf{B}^{loc}_{[\mathcal{C},int]}$.

Proof. (1) Let $alph \in \{inp, out, int\}$. Then $\mathbf{B}^{\Sigma}_{\mathcal{C}} = \mathbf{B}^{\Sigma}_{[\mathcal{C},alph]} = \mathrm{pres}_{\Sigma}(\mathbf{B}^{\Sigma}_{[\mathcal{C},alph]}) = \mathbf{B}^{alph}_{[\mathcal{C},alph]}$.

(2) $\mathbf{B}^{\Sigma}_{[\mathcal{C},inp]} = \mathrm{pres}_{\Sigma}(\mathbf{B}^{\Sigma}_{[\mathcal{C},inp]}) = \mathbf{B}^{inp}_{[\mathcal{C},inp]}$ and $\mathbf{B}^{ext}_{[\mathcal{C},inp]} = \mathrm{pres}_{\Sigma\cup\varnothing}(\mathbf{B}^{\Sigma}_{[\mathcal{C},inp]}) = \mathrm{pres}_{\Sigma}(\mathbf{B}^{\Sigma}_{[\mathcal{C},inp]})$.

(3,4) Analogous to (2). □

Now we denote $\mathsf{CA}^{alph} = \{\mathbf{B}^{alph}_{\mathcal{C}} \mid \mathcal{C}$ is a finite component automaton$\}$, with $alph \in \{inp, out, int, ext, loc\}$.

All languages in CA^{alph} are the images under a weak coding pres_{Σ} of languages in $\mathsf{CA} = \mathsf{pREG}$. It is known that pREG is closed under (weak) codings, i.e. whenever $L \in \mathsf{pREG}$ and L' is a (weak) coding of L, then we know that also $L' \in \mathsf{pREG}$. Using this closure of pREG under weak codings we immediately obtain the following result.

Lemma 6.1.3. *Let* $alph \in \{inp, out, int, ext, loc\}$. *Then*

$$\mathsf{CA}^{alph} \subseteq \mathsf{pREG}.$$ □

Combining this lemma with Lemmata 6.1.1 and 6.1.2 leads to the following result, which shows that the distinction of the set of actions into input, output, and internal actions has no influence on the behavior of finite component automata.

Theorem 6.1.4. $\mathsf{pREG} = \mathsf{CA} = \mathsf{CA}^{inp} = \mathsf{CA}^{out} = \mathsf{CA}^{int} = \mathsf{CA}^{ext} = \mathsf{CA}^{loc}$. □

6.2 Team Behavior Versus Component Behavior

For the remainder of this chapter all component automata (and thus all team automata) have a possibly infinite set of states and a possibly infinite set of actions. We investigate the relation between the computations and behavior of team automata on the one hand, and those of their constituting component automata on the other hand. Since we know that subteams of a team automaton can be viewed as components of (an iterated version of) that team automaton, it suffices to study the relation between team automata and their constituting component automata.

We first continue our study started in Section 4.2. Given the computations (behavior) of a team automaton we investigate how to extract the computations (behavior) of its constituting component automata. Later we change focus and investigate how to combine the given computations (behavior) of a composable system in such a way that the resulting computations (behavior) are those of a team automaton over that composable system.

Initially we consider team automata in which all actions are *ai*. In such a team automaton, in every synchronization on a given action always the same component automata participate. The results we obtain in this case form a satisfying picture. Consequently we move on to consider team automata with only *free* actions. In such a team automaton — although depending on the state a component (team) automaton is in — in every synchonization on a given action always only one component automaton participates. Also in this case we obtain interesting results. Finally, in a team automaton with only *si* actions, the participation of component automaton in synchronizations is fully state dependent. We argue that a drastically different approach is required to obtain results in this case.

Notation 9. *Also in this chapter we once more assume a fixed, but arbitrary and possibly infinite index set $\mathcal{I} \subseteq \mathbb{N}$, which we will use to index the component automata involved. For each $i \in \mathcal{I}$, we let $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$ be a fixed component automaton and we use Σ_i to denote its set of actions $\Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$. Moreover, we once more let $\mathcal{S} = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$ be a fixed composable system and we let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a fixed team automaton over $\mathcal{S}$. Furthermore, we use Σ to denote its set of actions $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$. Recall that $\mathcal{I} \subseteq \mathbb{N}$ implies that $\mathcal{I}$ is ordered by the usual $\leq$ relation on $\mathbb{N}$, thus inducing an ordering on $\mathcal{S}$, and that the $\mathcal{C}_i$ are not necessarily different. Finally, we let Θ be an arbitrary but fixed alphabet disjoint from Q.* □

6.2.1 From Team Automata to Component Automata

In this subsection we assume that the computations and behavior of a team automaton are given. From these we want to extract computations and behavior of its constituting component automata. We start by addressing this issue element-wise, i.e. given one particular computation (behavior) of a team automaton, we want to know whether we can extract from it the underlying computation (behavior) of one of its constituting component automata.

Notation 10. *For the remainder of this section we let $j \in \mathcal{I}$.* □

Given a team computation $\alpha \in \mathbf{C}^{\infty}_{\mathcal{T}}$ we know from Corollary 4.2.7 that $\pi_{\mathcal{C}_j}(\alpha) \in \mathbf{C}^{\infty}_{\mathcal{C}_j}$. Hence we can simply apply projections on the computations of team automata in order to obtain computations of its constituting component automata. Moreover, by definition, $\mathrm{pres}_{\Theta}(\alpha) \in \mathbf{B}^{\Theta,\infty}_{\mathcal{T}}$ and $\mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha)) \in \mathbf{B}^{\Theta,\infty}_{\mathcal{C}_j}$. This reflects the fact that behavior is obtained by filtering out state information from computations. We thus have the situation depicted by the diagram in Figure 6.1.

$$\begin{tikzcd}
\alpha \in \mathbf{C}^{\infty}_{\mathcal{T}} \arrow[r, "\pi_{\mathcal{C}_j}"] \arrow[d, "\mathrm{pres}_{\Theta}"'] & \pi_{\mathcal{C}_j}(\alpha) \in \mathbf{C}^{\infty}_{\mathcal{C}_j} \arrow[d, "\mathrm{pres}_{\Theta}"] \\
\mathrm{pres}_{\Theta}(\alpha) \in \mathbf{B}^{\Theta,\infty}_{\mathcal{T}} \arrow[r, dashed, "?"] & \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha)) \in \mathbf{B}^{\Theta,\infty}_{\mathcal{C}_j}
\end{tikzcd}$$

Fig. 6.1. Extracting behavior from team automata to component automata.

In addition we would like to obtain the Θ-behavior $\mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$ of component automaton $\mathcal{C}_j$ directly from the Θ-behavior $\mathrm{pres}_{\Theta}(\alpha)$ of team automaton $\mathcal{T}$. We thus look for an operation that makes the diagram of Figure 6.1 commute. A natural candidate is the homomorphism pres_{Σ_j} preserving only those actions from $\mathrm{pres}_{\Theta}(\alpha)$ that belong to component automaton $\mathcal{C}_j$. Hence we wonder whether $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$. In the following example we show that this equality in general does not hold.

Notation 11. *For the remainder of this section we may also specify our fixed component automata $\mathcal{C}_i$ as $(Q_i, \Sigma_i, \delta_i, I_i)$, $i \in \mathcal{I}$, and our fixed team automaton $\mathcal{T}$ as (Q, Σ, δ, I) whenever the distinctions of their alphabets into input, output, and internal actions are irrelevant.* □

Example 6.2.1. Let component automata $\mathcal{C}_1 = (\{q_1, q_1'\}, \{a, b\}, \{(q_1, b, q_1), (q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{C}_2 = (\{q_2, q_2'\}, \{a, b\}, \{(q_2, a, q_2'), (q_2', b, q_2')\}, \{q_2\})$ be as depicted in Figure 6.2.

We assume $\{\mathcal{C}_1, \mathcal{C}_2\}$ to be a composable system and consider team automaton $\mathcal{T} = (Q, \{a, b\}, \delta, \{(q_1, q_2)\})$, with $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$ and $\delta = \{((q_1, q_2), b, (q_1, q_2)), ((q_1, q_2), a, (q_1', q_2'))\}$, over this composable system. It is depicted in Figure 6.3(a).

Let $\alpha = (q_1, q_2) b (q_1, q_2) a (q_1', q_2') \in \mathbf{C}_{\mathcal{T}}$. Then $\mathrm{pres}_{\Sigma_2}(\mathrm{pres}_{\{a,b\}}(\alpha)) = ba \neq a = \mathrm{pres}_{\{a,b\}}(q_2 a q_2') = \mathrm{pres}_{\{a,b\}}(\pi_{\mathcal{C}_2}(\alpha))$. □

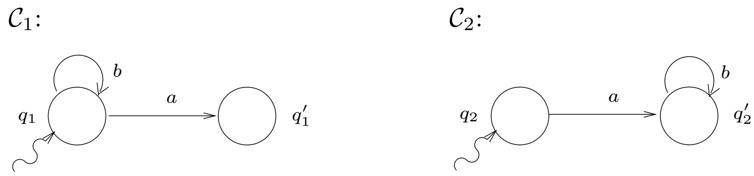


Fig. 6.2. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$.

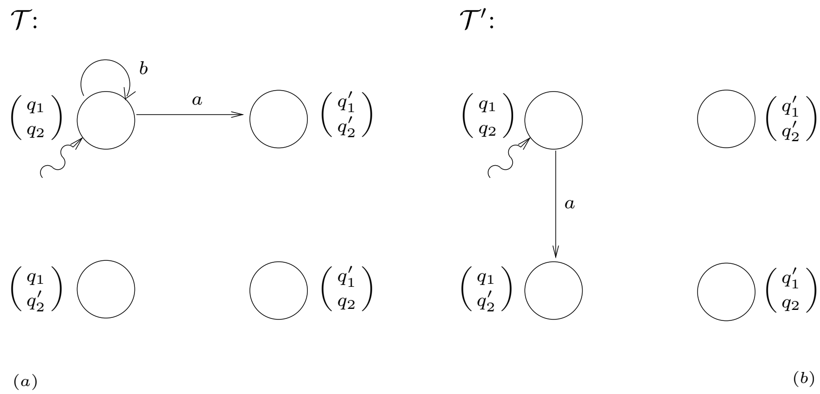


Fig. 6.3. Team automata $\mathcal{T}$ and $\mathcal{T}'$.

This example shows that in general we cannot assume that a component automaton participates in a synchronization, *just* because it has the action that is being synchronized as one of its actions. Hence there is no a priori relation between a component automaton's set of actions and its participation in synchronizations of those actions. The question we ask ourselves in this section now boils down to finding a necessary and sufficient condition which guarantees that $\text{pres}_{\Sigma_j}(\text{pres}_{\Theta}(\alpha)) = \text{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$.

As suggested by the example, we thus need to find a way to know whether or not a component automaton participates in a synchronization of the team automaton. It is therefore not surprising that the condition we present next is based on the *ai* principle, since every synchronization of an *ai* action involves all component automata that share this action. However, we obviously do not care about useless transitions as they can never be used anyway. It thus suffices to require the actions of $\mathcal{T}$ to be *ai* with respect to useful transitions only. Furthermore, for a given component $\mathcal{C}_j$ and action $a \in \Sigma_j$ it suffices to know that a is *ai with respect to* j, i.e. it is sufficient if $\mathcal{C}_j$ is required to participate in every useful a-transition of $\mathcal{T}$. This leads to the following definition.

Definition 6.2.2. *The set of* useful j-action-indispensable *actions is denoted by $uAI_j(\mathcal{T})$ and is defined as*

$$uAI_j(\mathcal{T}) = \{a \in \Sigma_j \mid \forall q, q' \in Q : (q, q') \in \delta_a \text{ is useful } \Rightarrow proj_j^{[2]}(q, q') \in \delta_{j,a}\}. \quad \square$$

Note that $AI(\mathcal{T}) \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$. We moreover note that whenever an action a of a component $\mathcal{C}_j$ is not active in $\mathcal{T}$, then $a \in uAI_j(\mathcal{T})$.

We can now formulate a sufficient condition under which the preserving homomorphism pres_{Σ_j} makes the diagram of Figure 6.1 commute. First we limit ourselves to finite computations.

Lemma 6.2.3. *If $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$, then for all $\alpha \in \mathbf{C}_{\mathcal{T}}$, $pres_{\Sigma_j}(pres_\Theta(\alpha)) = pres_\Theta(\pi_{\mathcal{C}_j}(\alpha))$.*

Proof. Let $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$ and let $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n \in \mathbf{C}_{\mathcal{T}}$. By induction on n we prove $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(\alpha)) = \text{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha))$.

If $n = 0$, then $\alpha = q_0$ and thus $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(q_0)) = \text{pres}_\Theta(\pi_{\mathcal{C}_j}(q_0)) = \lambda$.

Next assume that $n = k+1$, for some $k \geq 0$, and that $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(\beta)) = \text{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta))$, where $\beta = q_0 a_1 q_1 a_2 q_2 \cdots a_k q_k$. Hence $\alpha = \beta a_n q_n$. This implies that $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(\alpha)) = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\beta)) a_n$ if $a_n \in \Theta \cap \Sigma_j$ and $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(\alpha)) = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\beta))$ if $a_n \notin \Theta \cap \Sigma_j$.

First consider that $a_n \in \Theta \cap \Sigma_j$. Then $\text{proj}_j^{[2]}(q_n, q_{n+1}) \in \delta_{j,a_n}$ since $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$ and thus $\text{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \text{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta) a_n \text{proj}_j(q_{n+1})) = \text{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta)) a_n = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\beta a_n q_n))$ by the induction hypothesis. Hence $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(\alpha)) = \text{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha))$.

Next consider that $a_n \notin \Theta \cap \Sigma_j$. Then $a_n \notin \Theta$ or $a_n \notin \Sigma_j$.
If $a_n \notin \Sigma_j$, then $\pi_{\mathcal{C}_j}(\alpha) = \pi_{\mathcal{C}_j}(\beta)$ and thus, by the induction hypothesis, $\text{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \text{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta)) = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\beta))$. As $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(\beta)) = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\beta a_n q_n))$ it follows that $\text{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\alpha))$.
If $a_n \notin \Theta$, then $\text{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \text{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta))$ and thus, by the induction hypothesis, $\text{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\beta)) = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\alpha))$. $\square$

Next we allow also infinite computations.

Corollary 6.2.4. *If $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$, then for all $\alpha \in \mathbf{C}_{\mathcal{T}}^\infty$, $pres_{\Sigma_j}(pres_\Theta(\alpha)) = pres_\Theta(\pi_{\mathcal{C}_j}(\alpha))$.*

Proof. Let $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$ and let $\alpha \in \mathbf{C}_{\mathcal{T}}^\infty$. Due to Lemma 6.2.3 we only need to consider the infinite case. Hence we assume that $\alpha \in \mathbf{C}_{\mathcal{T}}^\omega$. Let $\alpha_1 \leq \alpha_2 \leq \cdots \in \mathbf{C}_{\mathcal{T}}$ be such that $\alpha = \lim_{n\to\infty} \alpha_n$. Thus $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(\alpha)) = \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\lim_{n\to\infty} \alpha_n))$. Then, by the definition of homomorphisms on infinite words, $\text{pres}_{\Sigma_j}(\text{pres}_\Theta(\lim_{n\to\infty} \alpha_n)) = \lim_{n\to\infty} \text{pres}_{\Sigma_j}(\text{pres}_\Theta(\alpha_n))$. Consequently, by the same reason and from Lemma 6.2.3 it now follows that $\lim_{n\to\infty} \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha_n)) = \lim_{n\to\infty} \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha_n)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\lim_{n\to\infty} \alpha_n)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$. □

It turns out that the condition proposed above is also necessary.

Lemma 6.2.5. *If $(\Theta \cap \Sigma_j) \setminus uAI_j(\mathcal{T}) \neq \varnothing$, then there exists an $\alpha \in \mathbf{C}_{\mathcal{T}}$ such that $pres_{\Sigma_j}(pres_{\Theta}(\alpha)) \neq pres_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$.*

Proof. Let $(\Theta \cap \Sigma_j) \setminus uAI_j(\mathcal{T}) \neq \varnothing$. Then the following situation must exist. Let $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n \in \mathbf{C}_{\mathcal{T}}$ be such that for all $1 \leq i < n$, either $a_i \notin \Theta$, or $a_i \notin \Sigma_j$, or $\mathrm{proj}_j{}^{[2]}(q_{i-1}, q_i) \in \delta_{j,a_i}$, while $\mathrm{proj}_j{}^{[2]}(q_{n-1}, q_n) \notin \delta_{j,a_n}$, with $a_n \in \Theta \cap \Sigma_j$. Hence $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(a_1 a_2 \cdots a_{n-1})) a_n$. Then $\mathrm{proj}_j{}^{[2]}(q_{n-1}, q_n) \notin \delta_{j,a_n}$ however implies that $\mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(q_0 a_1 q_1 a_2 q_2 \cdots a_{n-1} q_{n-1})) \neq \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(a_1 a_2 \cdots a_{n-1})) a_n = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha))$. □

We thus conclude that the proposed condition is necessary and sufficient for the diagram of Figure 6.1 to commute.

Theorem 6.2.6. *For all $\alpha \in \mathbf{C}^{\infty}_{\mathcal{T}}$, $pres_{\Sigma_j}(pres_{\Theta}(\alpha)) = pres_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$ if and only if $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$.*

Proof. (Only if) This is the contrapositive of Lemma 6.2.5.
(If) Directly from Corollary 6.2.4. □

Summarizing, we thus have the following situation. Whenever $\mathcal{C}_j$ contains at least one action from Θ which is not useful j-action-indispensable in $\mathcal{T}$, then $\mathcal{T}$ can execute a computation α for which $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha))$ does not equal $\mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$ (cf. Lemma 6.2.5).

Until now we extracted the behavior of the component automata of a team automaton from the computations of this team automaton. The above results however also provide us with a sufficient condition for obtaining the behavior of component automaton $\mathcal{C}_j$ directly from the behavior of team automaton $\mathcal{T}$, viz. by simply applying pres_{Σ_j} to its behavior.

Theorem 6.2.7. *If $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$, then $\mathbf{B}^{\Theta\cap\Sigma_j,\infty}_{\mathcal{T}} \subseteq \mathbf{B}^{\Theta,\infty}_{\mathcal{C}_j}$.*

Proof. Let $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$ and let $v \in \mathbf{B}^{\Theta\cap\Sigma_j,\infty}_{\mathcal{T}}$. This means that $v \in \mathrm{pres}_{\Theta\cap\Sigma_j}(\mathbf{C}^{\infty}_{\mathcal{T}})$. Now let $\alpha \in \mathbf{C}^{\infty}_{\mathcal{T}}$ be such that $\mathrm{pres}_{\Theta\cap\Sigma_j}(\alpha) = v$. From Corollary 4.2.7 we know that $\pi_{\mathcal{C}_j}(\alpha) \in \mathbf{C}^{\infty}_{\mathcal{C}_j}$. Since $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$, Corollary 6.2.4 implies that $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$. Hence $v = \mathrm{pres}_{\Theta\cap\Sigma_j}(\alpha) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha)) \in \mathbf{B}^{\Theta,\infty}_{\mathcal{C}_j}$. □

Note that Example 4.2.8 implies that it can still be the case that $\mathbf{B}_{\mathcal{T}}^{\Theta\cap\Sigma_j,\infty} \subset \mathbf{B}_{\mathcal{C}_j}^{\Theta,\infty}$, even if $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$.

Contrary to what might be expected from Theorem 6.2.6, the next example demonstrates that the statement from Theorem 6.2.7 cannot be reversed, i.e. $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$ is not a necessary condition for $\mathbf{B}_{\mathcal{T}}^{\Theta\cap\Sigma_j,\infty} \subseteq \mathbf{B}_{\mathcal{C}_j}^{\Theta,\infty}$ to hold. The reason is that the $\Theta \cap \Sigma_j$-behavior of $\mathcal{T}$ can be nonempty due to team computations in which $\mathcal{C}_j$ does not participate at all.

Example 6.2.8. (Example 6.2.1 continued) Consider team automaton $\mathcal{T}' = (Q, \{a,b\}, \{((q_1,q_2),a,(q_1,q_2'))\}, \{(q_1,q_2)\})$ over $\{\mathcal{C}_1,\mathcal{C}_2\}$. It is depicted in Figure 6.3(b).

Clearly $\mathbf{B}_{\mathcal{T}'}^{\Sigma,\infty} = \{\lambda, a\}$. Now let $\Theta = \{a,b\}$. Then $\Theta \cap \Sigma_1 = \{a,b\} \cap \{a,b\} = \{a,b\} \nsubseteq \{b\} = uAI_1(\mathcal{T}')$. However, $\mathbf{B}_{\mathcal{T}'}^{\Theta\cap\Sigma_1,\infty} = \mathbf{B}_{\mathcal{T}'}^{\{a,b\},\infty} = \{\lambda, a\} \subseteq \{b^n \mid n \geq 0\} \cup \{b^\omega\} \cup \{b^n a \mid n \geq 0\} = \mathbf{B}_{\mathcal{C}_1}^{\{a,b\},\infty} = \mathbf{B}_{\mathcal{C}_1}^{\Theta,\infty}$. □

Whereas a simple projection $\pi_{\mathcal{C}_j}$ applied to a computation of $\mathcal{T}$ suffices to obtain a computation of $\mathcal{C}_j$, a similarly simple preserving homomorphism pres_{Σ_j} applied to a behavior of $\mathcal{T}$ need not always yield a behavior of $\mathcal{C}_j$ *unless* all actions Σ_j of $\mathcal{C}_j$ are useful j-*ai*. The reason for this difference is as follows.

In a computation of $\mathcal{T}$ we still have available the information as to which components from $\mathcal{S}$ participated in each synchronization performed during this computation. When we deal with a behavior of $\mathcal{T}$, however, only the sequence of executed actions is available, i.e. we have lost all information as to which component automata from $\mathcal{S}$ participated in which execution. This implies that whenever we can be sure of a component automaton's participation in each execution of an action it has as an action itself, then we can simply apply our preserving homomorphism to the behavior of a team automaton in order to obtain the behavior of that component automaton.

Since every action of a component automaton from $\mathcal{S}$ is useful j-action-indispensable in the *maximal-ai* team automaton $\mathcal{T}$ over $\mathcal{S}$, Theorem 6.2.7 implies the following result. Slightly less general versions of this result, viz. without Θ being an arbitrary alphabet, have been formulated for other automata-based specification models with composition based on the *ai* principle (see, e.g., [Tut87] and [Jon87]). Theorems 6.2.6 and 6.2.7 however show a more precise condition guaranteeing this result and moreover exclude the existence of a similar relation in case composition is not based on the *ai* principle.

Theorem 6.2.9. *Let $\mathcal{T}$ be the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$. Then*

$$\mathbf{B}_{\mathcal{T}}^{\Theta\cap\Sigma_j,\infty} \subseteq \mathbf{B}_{\mathcal{C}_j}^{\Theta,\infty}.$$ □

At this point it is important to recall that in case $\mathcal{S}$ is such that none of its constituting component automata shares an action, then the *maximal-free* team automaton over $\mathcal{S}$ equals the *maximal-ai* team automaton over $\mathcal{S}$ (cf. Theorem 4.5.5) — in which case this theorem can thus be applied.

This completes our display of how to obtain the computations (behavior) of component automata constituting $\mathcal{S}$ from the computations (behavior) of team automata over $\mathcal{S}$. In the next section we study the dual approach.

6.2.2 From Component Automata to Team Automata

Contrary to the previous subsection we now assume that the computations and behavior of a set of component automata are given. Consequently we want to use this information to describe computations and behavior of team automata that can be composed over that set of component automata. We start by addressing this issue element-wise, i.e. given a computation (behavior) of each component automaton in a subset of $\mathcal{S}$ we want to know whether there exists a team automaton over $\mathcal{S}$ with a computation (behavior) that *uses* this combination of computations.

Definition 6.2.10. *Let* $\alpha \in \prod_{i\in\mathcal{I}} \mathbf{C}^{\infty}_{\mathcal{C}_i}$. *Then*

> α *is* used in $\mathcal{T}$ *if there exists a* $\beta \in \mathbf{C}^{\infty}_{\mathcal{T}}$ *such that for all* $i \in \mathcal{I}$, $\pi_{\mathcal{C}_i}(\beta) = proj_i(\alpha)$. □

Note that any vector of initial states is used in $\mathcal{T}$ since $\prod_{i\in\mathcal{I}} I_i \subseteq \mathbf{C}^{\infty}_{\mathcal{T}}$. If $K \subseteq \mathcal{I}$ and $\alpha_k \in \mathbf{C}^{\infty}_{\mathcal{C}_k}$, for all $k \in K$, then we say that $\prod_{k\in K} \alpha_k$ is *used in* $\mathcal{T}$ whenever there exists a $\gamma \in \prod_{i\in\mathcal{I}} \mathbf{C}^{\infty}_{\mathcal{C}_i}$ that is used in $\mathcal{T}$ and which is such that $\mathrm{proj}_k(\gamma) = \alpha_k$, for all $k \in K$. Finally, as vectors $\prod_{\{j\}} \mathbf{C}^{\infty}_{\mathcal{C}_j}$ have one element we will also say that $\alpha \in \mathbf{C}^{\infty}_{\mathcal{C}_j}$ is used in $\mathcal{T}$ whenever $\prod_{\{j\}} \alpha$ is.

In the following example we show that in general not all vectors over computations of component automata from $\mathcal{S}$ are used in $\mathcal{T}$. It may be the case that a computation of a component automaton from $\mathcal{S}$ never participates in a team computation. Moreover, it may happen that a vector over two or more computations of component automata from $\mathcal{S}$ is not used as such in $\mathcal{T}$, even when each entry of this vector *is* used in $\mathcal{T}$.

Example 6.2.11. (Examples 6.2.1 and 6.2.8 continued) We immediately see that $\mathcal{C}_2$ has a computation $\alpha' = q_2 a q_2' b q_2' \in \mathbf{C}_{\mathcal{C}_2}$ that is not used in $\mathcal{T}$ since there exists no $\beta \in \mathbf{C}^{\infty}_{\mathcal{T}}$ such that $\pi_{\mathcal{C}_2}(\beta) = \alpha'$.

Next we consider the team automaton $\mathcal{T}''$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$, which is obtained from team automaton $\mathcal{T}'$ as specified in Example 6.2.8 by adding transition $((q_1, q_2), a, (q_1', q_2))$ to its transition relation. It is depicted in Figure 6.4(a).

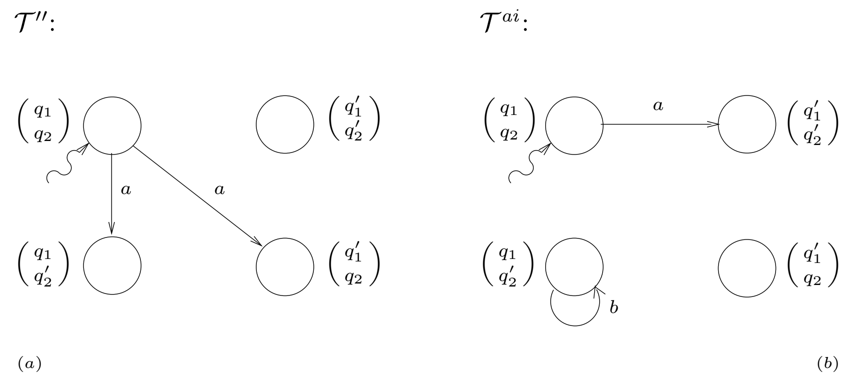


Fig. 6.4. Team automaton $\mathcal{T}''$ and *maximal-ai* team automaton $\mathcal{T}^{ai}$.

Clearly, both $\alpha_1 = q_1aq_1' \in \mathbf{C}_{\mathcal{C}_1}$ and $\alpha_2 = q_2aq_2' \in \mathbf{C}_{\mathcal{C}_2}$ are used in $\mathcal{T}''$ since $\beta_1 = (q_1, q_2)a(q_1', q_2) \in \mathbf{C}_{\mathcal{T}''}$ and $\beta_2 = (q_1, q_2)a(q_1, q_2') \in \mathbf{C}_{\mathcal{T}''}$ are such that $\pi_{\mathcal{C}_1}(\beta_1) = \alpha_1$ and $\pi_{\mathcal{C}_2}(\beta_2) = \alpha_2$. However, β_1 and β_2 are the only two nontrivial computations of $\mathcal{T}''$. Because $\pi_{\mathcal{C}_1}(\beta_2) = q_1$ and $\pi_{\mathcal{C}_2}(\beta_1) = q_2$ this means that there exists no $\beta \in \mathbf{C}^{\infty}_{\mathcal{T}''}$ such that $\pi_{\mathcal{C}_1}(\beta) = \alpha_1$ and $\pi_{\mathcal{C}_2}(\beta) = \alpha_2$. Hence (α_1, α_2) is not used in $\mathcal{T}''$.

Finally, note that (α_1, α_2) *is* used in team automaton $\mathcal{T}$ since $\beta = (q_1, q_2)a(q_1', q_2') \in \mathbf{C}_{\mathcal{T}}$ is such that $\pi_{\mathcal{C}_1}(\beta) = \mathrm{proj}_1((\alpha_1, \alpha_2)) = \alpha_1$ and $\pi_{\mathcal{C}_2}(\beta) = \mathrm{proj}_2((\alpha_1, \alpha_2)) = \alpha_2$. □

While in general not every vector over computations of component automata from $\mathcal{S}$ is used in $\mathcal{T}$, we wonder whether the situation improves in case $\mathcal{T}$ is defined in a particular way.

In analogy with the previous subsection, we first consider $\mathcal{T}$ to be such that all of its actions are *ai*. This is not yet enough, though, since whenever $\mathcal{T}$ has an empty transition relation, then all of its actions are *ai* while none of the computations of component automata from $\mathcal{S}$ is used in $\mathcal{T}$. Therefore we furthermore require $\mathcal{T}$ to be the *maximal-ai* team automaton over $\mathcal{S}$. However, in the following example we show that in general not all vectors over computations (behavior) of component automata from $\mathcal{S}$ are used in computations of the *maximal-ai* team automata over $\mathcal{S}$.

Example 6.2.12. (Example 6.2.11 continued) The *maximal-ai* team automaton over $\{\mathcal{C}_1, \mathcal{C}_2\}$ is $\mathcal{T}^{ai} = (Q, \{a, b\}, \delta^{ai}, \{(q_1, q_2)\})$, where $\delta^{ai} = \{((q_1, q_2), a, (q_1', q_2')), ((q_1, q_2'), b, (q_1, q_2'))\}$. It is depicted in Figure 6.4(b).

Trivially, $q_1 \in \mathbf{C}_{\mathcal{C}_1}$. However, since $(q_1, q_2)a(q_1', q_2')$ is the only nontrivial computation of $\mathcal{T}^{ai}$, there exists no computation $\beta' \in \mathbf{C}^{\infty}_{\mathcal{T}^{ai}}$ such that $\pi_{\mathcal{C}_1}(\beta') = q_1$ and $\pi_{\mathcal{C}_2}(\beta') = \alpha_2$. Hence (q_1, α_2) is not used in $\mathcal{T}^{ai}$. □

The fact that the *ai* type of synchronization forces component automata to synchronize on their shared actions provides us with enough information to formulate the conditions under which a vector of computations *is* used in a computation of the *maximal-ai* team automaton over $\mathcal{S}$. To this aim we define a vector α consisting of computations of the component automata from $\mathcal{S}$ — one for each such component automaton — to be *ai-consistent* if there exists a word w over Σ with the following property: whenever we preserve from w only the actions of a component automaton from $\mathcal{S}$, then we obtain exactly the behavior resulting from the computation in α that originates from that component automaton. In an *ai*-consistent vector the computations forming its entries thus "agree" with respect to the behavior of their respective components.

Definition 6.2.13. *Let* $\alpha \in \prod_{i \in \mathcal{I}} \mathbf{C}^{\infty}_{\mathcal{C}_i}$. *Then*

> α *is* ai-consistent *if there exists a* $w \in \Sigma^{\infty}$ *such that for all* $i \in \mathcal{I}$, $pres_{\Sigma_i}(w) = pres_{\Sigma_i}(proj_i(\alpha))$. □

It turns out that each *ai*-consistent vector over computations of component automata from $\mathcal{S}$ is used in the *maximal-ai* team automaton $\mathcal{T}$ over $\mathcal{S}$.

Lemma 6.2.14. *Let* $\mathcal{T}$ *be the* $\mathcal{R}^{ai}$*-team automaton over* $\mathcal{S}$ *and let* $\alpha \in \prod_{i \in \mathcal{I}} \mathbf{C}^{\infty}_{\mathcal{C}_i}$. *Then*

> *if* α *is ai-consistent, then* α *is used in* $\mathcal{T}$.

Proof. Let α be *ai*-consistent. Then by definition there exists a $w \in \Sigma^{\infty}$ such that $\mathrm{pres}_{\Sigma_i}(w) = \mathrm{pres}_{\Sigma_i}(\mathrm{proj}_i(\alpha))$, for all $i \in \mathcal{I}$.

First consider the case that $w \in \Sigma^*$. Let $w = a_1 a_2 \cdots a_n$ for some $n \geq 0$ and $a_k \in \Sigma$, for all $k \in [n]$. For each $i \in \mathcal{I}$, let the indices $i_1, i_2, \ldots, i_{n_i} \in [n]$ be such that $\mathrm{pres}_{\Sigma_i}(w) = a_{i_1} a_{i_2} \cdots a_{i_{n_i}}$. Hence $n_i = 0$ if $\mathrm{pres}_{\Sigma_i}(w) = \lambda$ and $1 \leq i_1 < i_2 < \cdots < i_{n_i} \leq n$ otherwise. Moreover, observe that $\bigcup_{i \in \mathcal{I}} \{i_1, i_2, \ldots, i_{n_i}\} = [n]$. Since for all $i \in \mathcal{I}$, $\mathrm{pres}_{\Sigma_i}(w) = \mathrm{pres}_{\Sigma_i}(\mathrm{proj}_i(\alpha))$ and $\mathrm{proj}_i(\alpha) \in \mathbf{C}_{\mathcal{C}_i}$, it follows that for all $i \in \mathcal{I}$, $\mathrm{proj}_i(\alpha) = q^i_0 a_{i_1} q^i_1 a_{i_2} \cdots a_{i_{n_i}} q^i_{n_i}$ with $q^i_0 \in I_i$ and $q^i_1, q^i_2, \ldots, q^i_{n_i} \in Q_i$.

Now define $\beta = q_0 a_1 q_1 a_2 \cdots a_n q_n$, with $q_k \in \prod_{i \in \mathcal{I}} Q_i$ for all $0 \leq k \leq n$, in such a way that for all $i \in \mathcal{I}$ and for all $0 \leq k \leq n$, $\mathrm{proj}_i(q_k) = q^i_{\ell}$ if $i_{\ell} \leq k < i_{\ell+1}$ with $\ell < n_i$ (by convention, $i_0 = 0$) and $\mathrm{proj}_i(q_k) = q^i_{n_i}$ if $i_{n_i} \leq k \leq n$. Consequently we prove that $\beta \in \mathbf{C}_{\mathcal{T}}$ while — in one stroke — $\pi_{\mathcal{C}_i}(\beta) = \mathrm{proj}_i(\alpha)$, for all $i \in \mathcal{I}$, follows from an inductive argument.

By its definition, $q_0 = \prod_{i \in \mathcal{I}} q^i_0 \in \prod_{i \in \mathcal{I}} I_i = I$. Next consider (q_{k-1}, a_k, q_k), for some $k \in [n]$. Let $i \in \mathcal{I}$. We distinguish the following two cases.

If $a_k \in \Sigma_i$, then $k = i_\ell$ for some $\ell \in [n_i]$ and $i_{\ell-1} \leq k-1 < k = i_\ell$. The definitions of q_{k-1} and q_k then yield $\mathrm{proj}_i(q_{k-1}) = q^i_{\ell-1}$ and $\mathrm{proj}_i(q_k) = q^i_\ell$. Since $\mathrm{proj}_i(\alpha) \in \mathbf{C}_{\mathcal{C}_i}$ it follows that $(q^i_{\ell-1}, q^i_\ell) \in \delta_{i,a_{i_\ell}} = \delta_{i,a_k}$.

If $a_k \notin \Sigma_i$, then $k \neq i_\ell$ for some $\ell \in [n_i]$.
If $k < i_{n_i}$, then there exists an $\ell \geq 1$ such that $i_{\ell-1} \leq k-1 < k < i_\ell$ and thus $\mathrm{proj}_i(q_{k-1}) = \mathrm{proj}_i(q_k) = q^i_{\ell-1}$.
Conversely, if $k \geq i_{n_i}$, then $i_{n_i} \leq k-1 < k \leq n$ and thus again $\mathrm{proj}_i(q_{k-1}) = \mathrm{proj}_i(q_k)$.

Since $\bigcup_{i \in \mathcal{I}} \{i_1, i_2, \ldots, i_{n_i}\} = [n]$, it follows that $a_k \in \Sigma_i$ for at least one $i \in \mathcal{I}$ and hence $(q_{k-1}, q_k) \in \mathcal{R}^{ai}_{a_k}(\mathcal{S}) = \delta_{a_k}$. This implies that for all $k \in [n]$, $q_0 a_1 q_1 a_2 \cdots a_k q_k \in \mathbf{C}_\mathcal{T}$ and for all $i \in \mathcal{I}$, $\pi_{\mathcal{C}_i}(q_0 a_1 q_1 a_2 \cdots a_k q_k) \in \mathbf{C}_{\mathcal{C}_i}$. Hence for all $i \in \mathcal{I}$, $\pi_{\mathcal{C}_i}(\beta) = \pi_{\mathcal{C}_i}(q_0 a_1 q_1 a_2 \cdots a_n q_n) = \mathrm{proj}_i(\alpha)$ and α is thus used in the *maximal-ai* team automaton $\mathcal{T}$.

Next consider the case that $w \in \Sigma^\omega$. Let $w = a_1 a_2 \cdots$, with $a_k \in \Sigma$ for all $k \geq 1$. Let $i \in \mathcal{I}$. For each $i \in \mathcal{I}$, if $\mathrm{pres}_{\Sigma_i}(w) \in \Sigma_i^*$, then as before there are indices $i_1, i_2, \ldots, i_{n_i}$ such that $\mathrm{pres}_{\Sigma_i}(w) = a_{i_1} a_{i_2} \cdots a_{i_{n_i}}$. Moreover, $\mathrm{proj}_i(\alpha) = q^i_0 a_{i_1} q^i_1 a_{i_2} \cdots a_{i_{n_i}} q^i_{n_i}$. If $\mathrm{pres}_{\Sigma_i}(w) \in \Sigma_i^\infty$, then there is an infinite sequence $1 \leq i_1 < i_2 < \cdots$ such that $\mathrm{pres}_{\Sigma_i}(w) = a_{i_1} a_{i_2} \cdots$. Then because w is such that for all $i \in \mathcal{I}$, $\mathrm{pres}_{\Sigma_i}(w) = \mathrm{pres}_{\Sigma_i}(\mathrm{proj}_i(\alpha))$, we can assume that $\mathrm{proj}_i(\alpha) = q^i_0 a_{i_1} q^i_1 a_{i_2} \cdots$ for some $q^i_k \in Q_i$, with $k \geq 0$.

Now we define $\beta = q_0 a_1 q_1 a_2 \cdots$ such that for all $i \in \mathcal{I}$, $\pi_{\mathcal{C}_i}(q_0) = q^i_0$ and $\pi_{\mathcal{C}_i}(q_k) = q^i_\ell$, for $i_\ell \leq k < i_{\ell+1}$ and $\ell \geq 1$. Clearly $q_0 \in I$. Similar to the finitary case it can now be shown that $(q_{k-1}, a_k, q_k) \in \delta$, for all $k \geq 1$.

Hence $\beta \in \mathbf{C}^\omega_\mathcal{T}$ and, moreover, $\pi_{\mathcal{C}_i}(\beta) = \mathrm{proj}_i(\alpha)$, for all $i \in \mathcal{I}$. □

From the proof of this lemma we immediately obtain the following result. Corresponding versions of both these results have been formulated for other automata-based specification models with composition based on the *ai* principle (see, e.g., [Tut87] and [Jon87]).

Corollary 6.2.15. *Let $\mathcal{T}$ be the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$ and let $\alpha \in \prod_{i \in \mathcal{I}} \mathbf{C}^\infty_{\mathcal{C}_i}$. Then*

if $w \in \Sigma^\infty$ is such that for all $i \in \mathcal{I}$, $pres_{\Sigma_i}(w) = pres_{\Sigma_i}(proj_i(\alpha))$, then there exists a $\beta \in \mathbf{C}^\infty_\mathcal{T}$ such that $pres_\Sigma(\beta) = w$. □

We thus see that *ai*-consistency is a sufficient condition for a vector over computations of component automata from $\mathcal{S}$ to be used in the *maximal-ai* team automaton over $\mathcal{S}$. Next we show that this condition is also necessary.

Theorem 6.2.16. *Let $\mathcal{T}$ be the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$ and let $\alpha \in \prod_{i \in \mathcal{I}} \mathbf{C}^\infty_{\mathcal{C}_i}$. Then*

α is used in $\mathcal{T}$ if and only if α is ai-consistent.

Proof. (If) This is Lemma 6.2.14.

(Only if) Let $\beta \in \mathbf{C}^{\infty}_{\mathcal{T}}$ be such that $\pi_{\mathcal{C}_i}(\beta) = \mathrm{proj}_i(\alpha)$, for all $i \in \mathcal{I}$. Since every action of $\mathcal{T}$ is *ai*, we can now apply Corollary 6.2.4 to obtain $\mathrm{pres}_{\Sigma_i}(\mathrm{pres}_{\Sigma}(\beta)) = \mathrm{pres}_{\Sigma}(\pi_{\mathcal{C}_i}(\beta)) = \mathrm{pres}_{\Sigma_i}(\pi_{\mathcal{C}_i}(\beta)) = \mathrm{pres}_{\Sigma_i}(\mathrm{proj}_i(\alpha))$, for all $i \in \mathcal{I}$. Hence α is *ai*-consistent. □

In order to formulate a general result relating the computations of *maximal-ai* team automata to the computations of their constituting component automata, we now define when a composable system is *ai*-consistent.

Definition 6.2.17. *$\mathcal{S}$ is* ai-consistent *if for all $i \in \mathcal{I}$ and for each $\gamma \in \mathbf{C}^{\infty}_{\mathcal{C}_i}$ there exists an ai-consistent vector $\alpha \in \prod_{i\in\mathcal{I}} \mathbf{C}^{\infty}_{\mathcal{C}_i}$ such that $proj_i(\alpha) = \gamma$.* □

Note that we have now defined *ai*-consistency both for vectors (over computations) and for composable systems. However, from the context it will always be clear whether we deal with an *ai*-consistent vector or rather with an *ai*-consistent composable system.

An *ai*-consistent composable system thus guarantees that for all computations of its constituents there exists a vector over such computations which is *ai*-consistent and thus each computation of a component automaton from $\mathcal{S}$ is used in a computation of the *maximal-ai* team automaton $\mathcal{T}$ over $\mathcal{S}$. In that case the set of computations (behavior) of a component automaton from $\mathcal{S}$ thus equals the set of computations (behavior) of the *maximal-ai* team automaton over $\mathcal{S}$ *projected* on that component automaton.

Theorem 6.2.18. *Let $\mathcal{T}$ be the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$. Then*

(1) $\mathbf{C}^{\infty}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}^{\infty}_{\mathcal{T}})$, for all $i \in \mathcal{I}$, if and only if $\mathcal{S}$ is ai-consistent, and

(2) if $\mathcal{S}$ is ai-consistent, then $\mathbf{B}^{\Sigma_i,\infty}_{\mathcal{C}_i} = \mathbf{B}^{\Sigma_i,\infty}_{\mathcal{T}}$, for all $i \in \mathcal{I}$.

Proof. (1) (Only if) Let $\mathbf{C}^{\infty}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}^{\infty}_{\mathcal{T}})$, for all $i \in \mathcal{I}$. Let $\gamma \in \mathbf{C}^{\infty}_{\mathcal{C}_k}$ for some $k \in \mathcal{I}$. Since $\mathbf{C}^{\infty}_{\mathcal{C}_k} = \pi_{\mathcal{C}_k}(\mathbf{C}^{\infty}_{\mathcal{T}})$ there exists a $\beta \in \mathbf{C}^{\infty}_{\mathcal{T}}$ such that $\pi_{\mathcal{C}_k}(\beta) = \gamma$. Now let $\alpha = \prod_{i\in\mathcal{I}} \pi_{\mathcal{C}_i}(\beta)$. Since $\pi_{\mathcal{C}_i}(\mathbf{C}^{\infty}_{\mathcal{T}}) = \mathbf{C}^{\infty}_{\mathcal{C}_i}$, for all $i \in \mathcal{I}$, it follows that $\alpha \in \prod_{i\in\mathcal{I}} \mathbf{C}^{\infty}_{\mathcal{C}_i}$. Furthermore, α is used and thus, by Theorem 6.2.16, α is *ai*-consistent. Definition 6.2.17 then implies that $\mathcal{S}$ is *ai*-consistent.

(If) Due to Corollary 4.2.7 we only need to prove that whenever $\mathcal{S}$ is *ai*-consistent, then for all $i \in \mathcal{I}$, $\mathbf{C}^{\infty}_{\mathcal{C}_i} \subseteq \pi_{\mathcal{C}_i}(\mathbf{C}^{\infty}_{\mathcal{T}})$. Now let $\gamma \in \mathbf{C}^{\infty}_{\mathcal{C}_k}$ for some $k \in \mathcal{I}$. Since $\mathcal{S}$ is *ai*-consistent there exists an *ai*-consistent vector $\alpha \in \prod_{i\in\mathcal{I}} \mathbf{C}^{\infty}_{\mathcal{C}_i}$ such that $\mathrm{proj}_k(\alpha) = \gamma$. Then by Theorem 6.2.16 there exists a $\beta \in \mathbf{C}^{\infty}_{\mathcal{T}}$ such that $\pi_{\mathcal{C}_k}(\beta) = \mathrm{proj}_k(\alpha) = \gamma$. Hence $\gamma \in \pi_{\mathcal{C}_k}(\mathbf{C}^{\infty}_{\mathcal{T}})$.

(2) Let $k \in \mathcal{I}$. Since $\mathcal{T}$ is the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$, Theorem 6.2.9 implies that $\mathbf{B}_{\mathcal{T}}^{\Sigma_k,\infty} \subseteq \mathbf{B}_{\mathcal{C}_k}^{\infty}$. Moreover, by (1) and Corollary 6.2.4, $\mathbf{B}_{\mathcal{C}_k}^{\infty} \subseteq \mathbf{B}_{\mathcal{T}}^{\Sigma_k,\infty}$. $\square$

Next we move on to the case that our team automaton $\mathcal{T}$ under consideration is the *maximal-free* team automaton over $\mathcal{S}$. Hence $\mathcal{T}$ consists of completely independent, non-synchronizing component automata. Consequently, our first intuition might be to jump to the conclusion that *every* single computation of a component automaton from $\mathcal{S}$ is used in $\mathcal{T}$.

As we have seen in Section 4.6, however, $\mathcal{T}$ does have one tricky characteristic in case loops are present in the component automata from $\mathcal{S}$: the combination of a loop, e.g. on a, in one component automaton from $\mathcal{S}$ and an a-transition in another component automaton from $\mathcal{S}$ results in the latter of these a-transitions not being omnipresent in $\mathcal{T}$. This implies that even if this a-transition is useful in its component automaton, i.e. it is part of a computation of that component automaton, then it is not at all guaranteed that this computation is used in $\mathcal{T}$. The reason for this is the maximal interpretation of the participation of transitions from component automata in transitions of team automata that we adopted in Section 4.2.

Indeed, in the following example we show that in general not each computation of a component automata from $\mathcal{S}$ is used in the *maximal-free* team automaton over $\mathcal{S}$.

Example 6.2.19. Let component automata $\mathcal{C} = (\{p\}, \{b\}, \{(p, b, p)\}, \{p\})$ and $\mathcal{C}' = (\{q, r\}, \{b\}, \{(q, b, r)\}, \{q\})$ be as depicted in Figure 6.5(a).

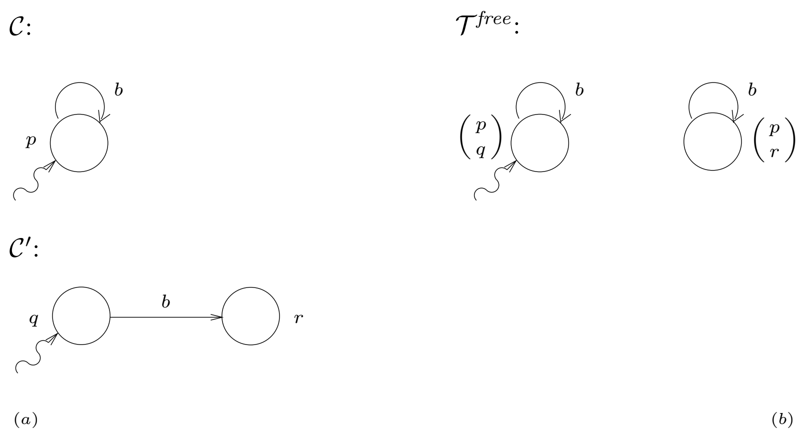


Fig. 6.5. Component automata $\mathcal{C}$ and $\mathcal{C}'$, and *maximal-free* team automaton $\mathcal{T}^{free}$.

Obviously $\{\mathcal{C}, \mathcal{C}'\}$ is a composable system. The $\mathcal{R}^{free}$-team automaton over $\{\mathcal{C}, \mathcal{C}'\}$ is $\mathcal{T}^{free} = (\{(p,q),(p,r)\}, \{b\}, \{((p,q),b,(p,q)), ((p,r),b,(p,r))\}, \{(p,q)\})$. It is depicted in Figure 6.5(b).

It is clear that $\alpha = qbr \in \mathbf{C}_{\mathcal{C}'}$ and that there does not exist a computation $\beta \in \mathbf{C}^{\infty}_{\mathcal{T}^{free}}$ such that $\pi_{\mathcal{C}'}(\beta) = \alpha$. Hence α is not used in $\mathcal{T}^{free}$. □

In case we only deal with a specific type of component automata, however, we can use Theorem 4.6.10(2). Recall that, given that $\mathcal{S}$ is j-loop limited and that $\mathcal{T}$ is the *maximal-free* team automaton over $\mathcal{S}$, this theorem states that every transition of $\mathcal{C}_j$ is omnipresent in $\mathcal{T}$. This means that whenever (p,a,p') is a transition of $\mathcal{C}_j$, then for all states q of $\mathcal{T}$ for which $\mathrm{proj}_j(q) = p$, there exists a transition (q,a,q') in $\mathcal{T}$ such that $\mathrm{proj}_j(q') = p'$, i.e. in which (p,a,p') is participating. Since $\mathcal{T}$ is the *maximal-free* team automaton over $\mathcal{S}$ we moreover know that $\mathrm{proj}_i(q') = \mathrm{proj}_i(q)$, for all $i \in \mathcal{I} \setminus \{j\}$, i.e. (p,a,p') is the only participating transition. It thus comes as no surprise that in case $\mathcal{S}$ is j-loop limited, each computation of a component automaton from $\mathcal{S}$ *is* used in a computation of the *maximal-free* team automaton over $\mathcal{S}$.

Lemma 6.2.20. *Let $\mathcal{T}$ be the $\mathcal{R}^{free}$-team automaton over $\mathcal{S}$ and let $\alpha \in \mathbf{C}^{\infty}_{\mathcal{C}_j}$. Then*

if $\mathcal{S}$ is j-loop limited, then α is used in $\mathcal{T}$.

Proof. Let $\mathcal{S}$ be j-loop limited.

First consider the case that $\alpha \in \mathbf{C}_{\mathcal{C}_j}$. Let $\alpha = p_0 a_1 p_1 a_2 \cdots a_n p_n$ for some $n \geq 0$, i.e. $(p_{k-1}, p_k) \in \delta_{j,a_k}$, for all $1 \leq k \leq n$. Since $Q = \prod_{i \in \mathcal{I}} Q_i$ and $I = \prod_{i \in \mathcal{I}} I_i$, Theorem 4.6.10(2) implies that there exists a computation $\beta = q_0 a_1 q_1 a_2 \cdots a_n q_n \in \mathbf{C}_{\mathcal{T}}$ such that $\mathrm{proj}_j{}^{[2]}(q_{k-1}, q_k) = (p_{k-1}, p_k) \in \delta_{j,a_k}$, for all $1 \leq k \leq n$. Hence $\pi_{\mathcal{C}_j}(\beta) = \alpha$ and α is thus used in $\mathcal{T}$.

Secondly, the case that $\alpha \in \mathbf{C}^{\omega}_{\mathcal{C}_j}$ is analogous to the finitary case. □

We thus see that loop limitedness is a sufficient condition for a vector over computations of component automata from $\mathcal{S}$ to be used in the *maximal-free* team automaton over $\mathcal{S}$. We will soon see that this condition is not necessary.

From Corollary 4.2.7 we know that given a computation of a team automaton $\mathcal{T}$ over $\mathcal{S}$, the projection on a component automaton from $\mathcal{S}$ is included in the set of computations of that component automaton. Together with Lemma 6.2.20 this implies that whenever $\mathcal{S}$ is j-loop limited, then the set of computations of a component automaton from $\mathcal{S}$ equals the set of computations of the *maximal-free* team automaton $\mathcal{T}$ over $\mathcal{S}$ *projected* on that component automaton. Moreover, the behavior of that component automaton is included in the behavior of $\mathcal{T}$. Like the proof of Lemma 6.2.20, also the proof of this statement is based on the observation that in a *maximal-free* team automaton, each executed action has only one participating component automaton. This implies that the team automaton can always execute any computation of any of its constituting component automata while keeping all remaining component automata from $\mathcal{S}$ in an initial state.

Theorem 6.2.21. *Let $\mathcal{T}$ be the $\mathcal{R}^{free}$-team automaton over $\mathcal{S}$. Then*

if $\mathcal{S}$ is loop limited, then $\mathbf{C}^{\infty}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}^{\infty}_{\mathcal{T}})$ and $\mathbf{B}^{\Sigma_i,\infty}_{\mathcal{C}_i} \subseteq \mathbf{B}^{\Sigma,\infty}_{\mathcal{T}}$, for all $i \in \mathcal{I}$.

Proof. Let $\mathcal{S}$ be loop limited. Then Lemma 6.2.20 implies that $\mathbf{C}^{\infty}_{\mathcal{C}_i} \subseteq \pi_{\mathcal{C}_i}(\mathbf{C}^{\infty}_{\mathcal{T}})$ and thus, by Corollary 4.2.7, $\mathbf{C}^{\infty}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}^{\infty}_{\mathcal{T}})$. Now let $k \in \mathcal{I}$, let $\alpha \in \mathbf{B}^{\Sigma_k,\infty}_{\mathcal{C}_k}$ and let $\beta \in \mathbf{C}^{\infty}_{\mathcal{C}_k}$ be such that $\mathrm{pres}_{\Sigma_k}(\beta) = \alpha$. Since $\mathbf{C}^{\infty}_{\mathcal{C}_k} = \pi_{\mathcal{C}_k}(\mathbf{C}^{\infty}_{\mathcal{T}})$, there must exist a $\gamma \in \mathbf{C}^{\infty}_{\mathcal{T}}$ such that $\beta = \pi_{\mathcal{C}_k}(\gamma)$. Moreover, since by Theorem 4.6.10(2) all transitions of $\mathcal{C}_k$ are omnipresent in $\mathcal{T}$, it follows that we may assume that $\pi_{\mathcal{C}_\ell}(\gamma) \in I_\ell$, for all $\ell \in \mathcal{I} \setminus \{k\}$. Hence $\mathrm{pres}_{\Sigma}(\gamma) = \mathrm{pres}_{\Sigma}(\pi_{\mathcal{C}_k}(\gamma)) = \mathrm{pres}_{\Sigma_k}(\beta) = \alpha$ and thus $\alpha \in \mathbf{B}^{\Sigma,\infty}_{\mathcal{T}}$. □

The behavior of the *maximal-free* team automaton $\mathcal{T}$ over $\mathcal{S}$ trivially is made up of the behavior of not just one component automaton from $\mathcal{S}$, but of the behavior of all of the component automata from $\mathcal{S}$. Therefore, in general $\mathbf{B}^{\Sigma_j,\infty}_{\mathcal{C}_j} \subseteq \mathbf{B}^{\Sigma,\infty}_{\mathcal{T}}$ will be proper, even if $\mathcal{S}$ is j-loop limited. Furthermore, the fact that $\mathbf{C}^{\infty}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}^{\infty}_{\mathcal{T}})$, for all $i \in \mathcal{I}$, need not imply that $\mathcal{S}$ is loop limited.

Example 6.2.22. (Example 6.2.11 continued) The *maximal-free* team automaton over $\{\mathcal{C}_1, \mathcal{C}_2\}$ is $\mathcal{T}^{free} = (\, Q,\ \{\, a,\, b\,\},\ \delta^{free},\ \{\,(q_1,\, q_2)\,\}\,)$, where $\delta^{free} = \{\,(\,(q_1, q_2), b, (q_1, q_2)\,),\ (\,(q_1, q_2), a, (q_1, q_2')\,),\ (\,(q_1, q_2), a, (q_1', q_2)\,),\ (\,(q_1, q_2'), a, (q_1', q_2')\,),\ (\,(q_1', q_2), a, (q_1', q_2')\,),\ (\,(q_1', q_2'), b, (q_1', q_2')\,)\,\}$. It is depicted in Figure 6.6(a).

Since $\beta = (q_1, q_2)a(q_1, q_2')a(q_1', q_2')b(q_1', q_2') \in \mathbf{C}_{\mathcal{T}^{free}}$, α' is used in $\mathcal{T}^{free}$. It is moreover not difficult to see that for all $k \in [2]$, $\mathbf{C}^{\infty}_{\mathcal{C}_k} \subseteq \pi_{\mathcal{C}_k}(\mathbf{C}^{\infty}_{\mathcal{T}^{free}})$ and thus, by Corollary 4.2.7, $\mathbf{C}^{\infty}_{\mathcal{C}_k} = \pi_{\mathcal{C}_k}(\mathbf{C}^{\infty}_{\mathcal{T}^{free}})$. However, $\{\mathcal{C}_1, \mathcal{C}_2\}$ is not loop limited because $(q_1, q_1) \in \delta_{1,b}$ and $(q_2', q_2') \in \delta_{2,b}$. □

Note that Theorem 6.2.21 relies heavily on the fact that in the *maximal-free* team automaton over a loop-limited $\mathcal{S}$, each action of a component automaton can be executed independently of the current local states that the other component automata of $\mathcal{S}$ are in, since none of these other component automata participates in such an execution. In the *maximal-ai* team automaton over $\mathcal{S}$, this situation can only occur when none of the other component automata from $\mathcal{S}$ contains any of the actions that was executed as part of the computation of the *maximal-ai* team automaton. Hence even when $\mathcal{S}$ is *ai*-consistent, then the behavior of $\mathcal{C}_j$ is in general not contained in the behavior

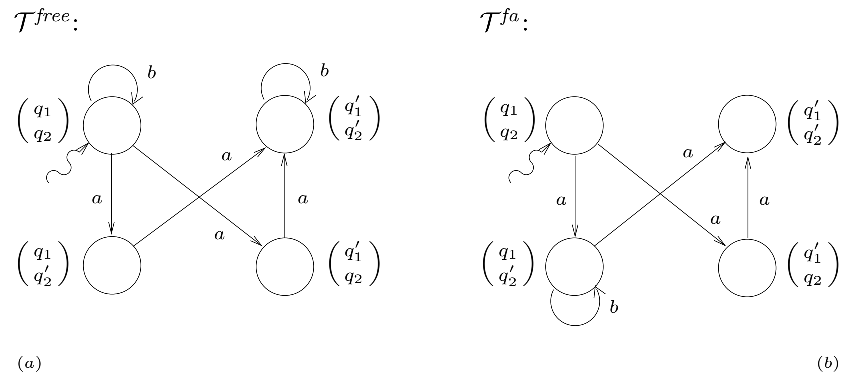


Fig. 6.6. Team automata $\mathcal{T}^{free}$ and $\mathcal{T}^{fa}$.

of the *maximal-ai* team automaton over $\mathcal{S}$. From Theorem 6.2.18(2) we however know that if $\mathcal{S}$ is *ai*-consistent, then the behavior of $\mathcal{C}_j$ *is* contained in the Σ_j-behavior of the *maximal-ai* team automaton over $\mathcal{S}$.

Both for *maximal-ai* team automata (cf. Theorem 6.2.18(1)) and for *maximal-free* team automata (cf. Theorem 6.2.21) over $\mathcal{S}$ we have thus found a sufficient condition on $\mathcal{S}$ (*ai*-consistency and loop limitedness, respectively) under which all component computations contribute to team computations. In case of *maximal-ai* team automata this condition is even necessary. As direct consequences of these results we have subsequently been able to relate the behavior of component automata to that of *maximal-ai* team automata (cf. Theorem 6.2.18(2)) and to that of *maximal-free* team automata (cf. Theorem 6.2.21).

In the remainder of this chapter we moreover define the behavior of the *maximal-ai* (*maximal-free*) team automaton over $\mathcal{S}$ in terms of the behavior of its constituting component automata. This requires establishing which combinations of words — if any — from the behavior of component automata from $\mathcal{S}$ can be combined — and in particular how — such that a word from the behavior of the *maximal-ai* (*maximal-free*) team automaton over $\mathcal{S}$ results. For this we use *shuffling* operations, known from the theory of formal languages. We will consider both "free" shuffles (to deal with *free* actions) and "synchronized" shuffles (to deal with *ai* actions).

In the succeeding two sections we formally define the different kinds of shuffles and study some of their properties. In the subsequent and final section of this chapter we then show that the behavior of team automata constructed on the basis of *maximal-ai* and/or *maximal-free* synchronizations can be expressed as a (synchronized) shuffle of the behavior of their constituting component automata, where the kind of shuffle depends on the type of synchronization.

The two sections dealing with various kinds of shuffles are rather technical and relatively extensive. One of the main reasons for this resides in the fact that our team automata framework allows for infinite computations and infinite behavior. Therefore we need to consider shuffles on finite as well as infinite words. Moreover, when dealing with composable systems consisting of two or more component automata, notions of commutativity and associativity for the various kinds of shuffles are of crucial importance. Readers interested only in the results can jump to the final section of this chapter and when necessary skim Subsections 6.3.1, 6.3.4, 6.4.1, and 6.4.4 for the definitions needed to interpret the results.

6.3 Shuffles

This section marks the beginning of our exposition on shuffles. The idea behind a shuffle of languages is the creation of a new language, the words of which consist of the words of the original languages "woven" in a particular fashion. For one, words of the original languages are part of the words of the new language. Consider, e.g., the (finite) words eae and wv. Then we can weave these words into a new (finite) word $weave$. To the best of our knowledge, the oldest reference to this way of shuffling two (finite) words is [GS65], which was presented at a conference as early as 1964.

In this simple example we described a shuffle of two finite words. We know, however, that the languages of our component (team) automata may contain infinite words. When we try to shuffle two infinite words in the above-mentioned way we are forced to take some decisions concerning "fairness". Consider, e.g., the words a^ω and b^ω. Then we can weave these words into new (infinite) words of the form $(a^+b^+)^\omega$, consisting of both infinitely many a's and infinitely many b's. Hence a^ω and b^ω are woven in a fair way: finite nonempty subwords of the two words occur alternatingly, i.e. each word gets its fair turn in the new words. However, we could also decide to allow infinite subwords of the original words to appear in the new word. In that case a result of weaving a^ω and b^ω can be an (infinite) word from $(a^+b^+)^*a^\omega$. Note, however, that in this case the result does not contain an infinite number of b's. The oldest reference — again, to the best of our knowledge — to this idea of shuffling two infinite words is [Sha78], and to this idea of fair shuffling is [Par79] (where fair shuffling is called fair merging, though).

These simplified examples suggest that there is a clear need to define precisely and unambiguously what types of shuffles we shall consider. Another reason for being more precise is to avoid the confusion that could arise from the fact that the (fair) shuffle is a well-known language-theoretic operation. It thus has a long history within theoretical computer science, in particular within formal language theory. Shuffling is sometimes called interleaving, weaving, or merging, and — given two words u and v — it may be denoted by $u \odot v$, $u \mid\mid v$, $u \sqcup\!\sqcup v$, $u \,\Box\, v$, $u \otimes v$, $u \,|||\, v$, or $u \diamond v$ (see, e.g., [GS65], [Sha78], [Par79], [Gis81], [Jan81], [BÉ96], [RS97]). The idea of shuffling also appears in numerous other disguises throughout the computer science literature. Within concurrency theory, e.g., as a semantics of parallel operators modeling communication between processes (see, e.g., [Ros97] and [BPS01]). In the next section we will consider also shuffles which are not merely interleavings, but which may require the synchronized occurrence of certain symbols.

The remainder of this section and the subsequent section together form a self-contained theory of shuffles. By no means do we claim that all results are new. We are aware of the fact that some results have appeared in the literature, but we have been unable to find a comprehensive theory of shuffles in the literature that would suit our approach. Due to the generic setup of the team automaton model we need to be able to deal with shuffles of infinite words and, moreover, we need several specific shuffles that are combinations of shuffling and synchronizing. Most of this has largely gone unexplored in the literature.

In this section we introduce the basic shuffle, well-known from the literature. We study its basic properties and prove its commutativity and associativity. In the subsequent section we consequently introduce three more intricate types of shuffles, built on top of the basic shuffle. We briefly study also their properties and establish notions of commutativity and associativity also for these types of shuffles. The fact that all four shuffles satisfy some sort of commutativity and associativity is crucial for applying them in the context of team automata in the final section of this chapter.

6.3.1 Definitions

We begin by introducing the basic shuffle.

Definition 6.3.1. *Let $u, v \in \Delta^\infty$. Then*

a word $w \in \Delta^\infty$ is a shuffle *of u and v if one of the following four cases holds. Either*

(1) $u, v \in \Delta^*$ and $w = u_1 v_1 u_2 v_2 \cdots u_n v_n$, where $n \geq 1$, $u_1 \in \Delta^*$, $u_2, u_3, \ldots, u_n, v_1, v_2, \ldots, v_{n-1} \in \Delta^+$, $v_n \in \Delta^*$, $u_1 u_2 \cdots u_n = u$, and $v_1 v_2 \cdots v_n = v$, or

(2) $u \in \Delta^*$, $v \in \Delta^\omega$, *and* $w = u_1 v_1 u_2 v_2 \cdots u_n v_n$, *where* $n \geq 1$, $u_1 \in \Delta^*$, $u_2, u_3, \ldots, u_n, v_1, v_2, \ldots, v_{n-1} \in \Delta^+$, $v_n \in \Delta^\omega$, $u_1 u_2 \cdots u_n \in \mathit{pref}(u)$, *and* $v_1 v_2 \cdots v_n = v$, *or*

(3) $u \in \Delta^\omega$, $v \in \Delta^*$, *and* w *is a shuffle of* v *and* u, *or*

(4) $u, v \in \Delta^\omega$ *and either*
  - *(a)* w *is a shuffle of* u *and a prefix of* v, *or*
  - *(b)* w *is a shuffle of* v *and a prefix of* u, *or*
  - *(c)* $w = u_1 v_1 u_2 v_2 \cdots$, *where* $u_1 \in \Delta^*$, $u_j, v_1, v_j \in \Delta^+$ *for all* $j \geq 2$, $u = \lim_{n\to\infty} u_1 u_2 \cdots u_n$, *and* $v = \lim_{n\to\infty} v_1 v_2 \cdots v_n$.

A shuffle w *of* u *and* v *is called* fair *(w.r.t.* u *and* v*) if* u *and* v *are finite (case (1)), or if in case (2)* $u_1 u_2 \cdots u_n = u$, *or if in case (3)* w *is a fair shuffle of* v *and* u, *or if in case (4) subcase (c) holds.* □

For $u, v \in \Delta^\infty$ the language consisting of all *(fair) shuffles* of u and v is denoted by $u \mathbin{||} v$ ($u \mathbin{|||} v$) and is defined as $u \mathbin{||} v = \{w \in \Delta^\infty \mid w$ is a shuffle of u and $v\}$ and $u \mathbin{|||} v = \{w \in \Delta^\infty \mid w$ is a fair shuffle of u and $v\}$, respectively.

For $L_1, L_2 \subseteq \Delta^\infty$ the *(fair) shuffle* of L_1 and L_2 is denoted by $L_1 \mathbin{||} L_2$ ($L_1 \mathbin{|||} L_2$) and is defined as the language consisting of all words which are a (fair) shuffle of a word from L_1 and a word from L_2. Thus $L_1 \mathbin{||} L_2 = \{w \in u \mathbin{||} v \mid u \in L_1,\ v \in L_2\} = \bigcup_{u \in L_1,\, v \in L_2} (u \mathbin{||} v)$ and $L_1 \mathbin{|||} L_2 = \bigcup_{u \in L_1,\, v \in L_2} (u \mathbin{|||} v)$.

Example 6.3.2. Let $\Delta = \{a, b, c, d\}$. Let $u = abc \in \Delta^*$ and let $v = cd \in \Delta^*$. Then $u \mathbin{||} v = \{\mathit{abccd}, \mathit{acbcd}, \mathit{cabcd}, \mathit{abcdc}, \mathit{acbdc}, \mathit{cabdc}, \mathit{acdbc}, \mathit{cadbc}, \mathit{cdabc}\} = u \mathbin{|||} v$.

Consequently, let $w_1 = a^\omega \in \Delta^\infty$ and let $w_2 = b^\omega \in \Delta^\infty$. Then $(ab)^\omega$ is a fair shuffle of w_1 and w_2, whereas aba^ω is a shuffle of w_1 and w_2, but not a fair shuffle.

Moreover, note that $v \mathbin{|||} w_2 = \{b^m c b^n d b^\omega \mid m, n \geq 0\}$, whereas $v \mathbin{||} w_2 = \{b^\omega\} \cup \{b^n c b^\omega \mid n \geq 0\} \cup v \mathbin{|||} w_2$. □

6.3.2 Basic Observations

Definition 6.3.1 immediately implies that the fair shuffle of two languages is included in the shuffle of those two languages.

Lemma 6.3.3. *Let* $u, v \in \Delta^\infty$ *and let* $L_1, L_2 \subseteq \Delta^\infty$. *Then*

(1) $u \mathbin{|||} v \subseteq u \mathbin{||} v$ *and*

(2) $L_1 \mid\mid\mid L_2 \subseteq L_1 \mid\mid L_2$. □

From Example 6.3.2 we conclude that both of these inclusions may be proper. In fact, the following result follows immediately from Definition 6.3.1.

Lemma 6.3.4. *(1) If $u, v \in \Delta^*$, then $u \mid\mid v = u \mid\mid\mid v$,*

(2) if $u \in \Delta^*$ and $v \in \Delta^\omega$, then $u \mid\mid v = \bigcup_{u' \in \mathit{pref}(u)} (u' \mid\mid\mid v)$, and

(3) if $u, v \in \Delta^\omega$, then $u \mid\mid v = \bigcup_{u' \in \mathit{pref}(u)} (u' \mid\mid\mid v) \cup \bigcup_{v' \in \mathit{pref}(v)} (u \mid\mid\mid v') \cup u \mid\mid\mid v$. □

Example 6.3.5. (Example 6.3.2 continued) We thus have that $w_1 \mid\mid w_2 = (a^* \mid\mid\mid \{w_2\}) \cup (\{w_1\} \mid\mid\mid b^*) \cup (w_1 \mid\mid\mid w_2)$, with $w_1 \mid\mid\mid w_2 = (a^+ \mid\mid b^+)^\omega$. □

Note furthermore that two words always define at least one (fair) shuffle, i.e. given $u, v \in \Delta^\infty$, then $u \mid\mid\mid v \neq \varnothing$ (and thus $u \mid\mid v \neq \varnothing$). Whenever both u and v equal λ, however, then $u \mid\mid v = u \mid\mid\mid v = \{\lambda\}$. Also the case that only one of the words u and v is λ exhibits no surprises.

Lemma 6.3.6. *Let $u \in \Delta^\infty$. Then*

$$u \mid\mid \lambda = u \mid\mid\mid \lambda = \{u\} = \lambda \mid\mid\mid u = \lambda \mid\mid u.$$ □

In Definition 6.3.1 we have defined a (fair) shuffle of two words as an (infinite) alternation of (finite) nonempty subwords of the one word with (finite) nonempty subwords of the other word. We now show that dropping the requirement that the subwords be nonempty does not alter the definition.

Lemma 6.3.7. *Let $u, v \in \Delta^\infty$. Then*

(1) $w \in u \mid\mid\mid v$ if and only if $w = u_1 v_1 u_2 v_2 \cdots$, with $u_i, v_i \in \Delta^*$ for all $i \geq 1$, $u = u_1 u_2 \cdots$, and $v = v_1 v_2 \cdots$, and

(2) $w \in u \mid\mid v$ if and only if $w \in u \mid\mid\mid v$ or $w = u_1 v_1 u_2 v_2 \cdots$, with $u_i, v_i \in \Delta^*$ for all $i \geq 1$, and either $u_1 u_2 \cdots \in \mathit{pref}(u)$ and $v = v_1 v_2 \cdots$ or $u = u_1 u_2 \cdots$ and $v_1 v_2 \cdots \in \mathit{pref}(v)$.

Proof. (1) (Only if) Immediate from Definition 6.3.1.

(If) Let $w = u_1 v_1 u_2 v_2 \cdots$, with $u_i, v_i \in \Delta^*$ for all $i \geq 1$, $u = u_1 u_2 \cdots$, and $v = v_1 v_2 \cdots$. The proof of the statement makes use of the following construction, which provides representations ρ_k, $k \geq 1$, of prefixes of w satisfying some particular properties. Formally, the representations ρ_k, for all $k \geq 1$, are defined by $\rho_1 = (u_1, v_1)$ and if $\rho_k = (\alpha_1, \beta_1, \alpha_2, \beta_2, \cdots, \alpha_\ell, \beta_\ell)$ for some $l \geq 1$ and $\alpha_j, \beta_j \in \Delta^*$, for all $1 \leq j \leq \ell$, then

$$\rho_{k+1} = \begin{cases} (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell u_{k+1}, v_{k+1}) & \text{if } \beta_\ell = \lambda, \\ (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell v_{k+1}) & \text{if } \beta_\ell \neq \lambda \text{ and } u_{k+1} = \lambda, \text{ and} \\ (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell, u_{k+1}, v_{k+1}) & \text{if } \beta_\ell \neq \lambda \text{ and } u_{k+1} \neq \lambda. \end{cases}$$

The representation ρ_{k+1} is thus obtained by adding the words u_{k+1} and v_{k+1} to ρ_k. They are added to ρ_k in such a way that only the first and the last element of ρ_{k+1} are allowed to equal λ. As a result in the representation ρ_{k+1} of the prefix $u_1 v_1 u_2 v_2 \cdots u_{k+1} v_{k+1}$ all intermediate λ's have been omitted. Formally, the representations thus constructed possess the following properties that we use in this proof. Let $\rho_k = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell)$ for some $l \geq 1$ and $\alpha_j, \beta_j \in \Delta^*$, for all $j \in [\ell]$. Then $\alpha_1, \beta_\ell \in \Delta^*$, $\alpha_j \in \Delta^+$, for all $1 < j \leq \ell$, and $\beta_j \in \Delta^+$, for all $1 \leq j < \ell$. Furthermore, $\alpha_1 \beta_1 \alpha_2 \beta_2 \cdots \alpha_\ell \beta_\ell = u_1 v_1 u_2 v_2 \cdots u_k v_k$, $\alpha_1 \alpha_2 \cdots \alpha_\ell = u_1 u_2 \cdots u_k$, and $\beta_1 \beta_2 \cdots \beta_\ell = v_1 v_2 \cdots v_k$. We now turn to the actual proof.

First consider the case that $u, v \in \Delta^*$. Since w is an infinite alternation of $u_i, v_i \in \Delta^*$, there must exist an $m \geq 1$ such that for all $n > m$, $u_n = v_n = \lambda$. Then $\rho_m = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell)$ is such that $\alpha_1 \beta_1 \alpha_2 \beta_2 \cdots \alpha_\ell \beta_\ell = w$, $\alpha_1, \beta_\ell \in \Delta^*$, and $\beta_1, \alpha_2, \beta_2, \alpha_3, \ldots, \beta_{\ell-1}, \alpha_\ell \in \Delta^+$. Hence $w \in u \mathbin{|||} v$.

Next consider the case that $u \in \Delta^*$ and $v \in \Delta^\omega$. Hence there must exist an $m \geq 1$ such that for all $n > m$, $u_n = \lambda$. Then with $\rho_m = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell)$ we obtain that for all $k \geq 1$, $\rho_{m+k} = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell v_{m+1} v_{m+2} \cdots v_{m+k})$, where $\alpha_1, \beta_\ell v_{m+1} v_{m+2} \cdots v_{m+k} \in \Delta^*$, $\alpha_j \in \Delta^+$, for all $1 < j \leq \ell$, $\beta_j \in \Delta^+$, for all $1 \leq j < \ell$, and $w = \alpha_1 \beta_1 \alpha_2 \beta_2 \cdots \alpha_\ell \beta_\ell v_{m+1} v_{m+2} \cdots$. Hence $w \in u \mathbin{|||} v$.

Now consider the case that $u \in \Delta^\omega$ and $v \in \Delta^*$. Let $m \geq 1$ be the smallest index such that $u_m \neq \lambda$ and for all $n \geq m$, $v_n = \lambda$. Then with $\rho_m = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell)$, where $\beta_\ell = \lambda$ we obtain that for all $k \geq 1$, $\rho_{m+k} = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell u_{m+1} u_{m+2} \cdots u_{m+k}, \lambda)$, where $\alpha_1 \in \Delta^*$, $\alpha_j \in \Delta^+$, for all $1 < j < \ell$, $\alpha_\ell u_{m+1} u_{m+2} \cdots u_{m+k} \in \Delta^+$, $\beta_j \in \Delta^+$, for all $1 \leq j < \ell$, and $w = \alpha_1 \beta_1 \alpha_2 \beta_2 \cdots \beta_{\ell-1} \alpha_\ell u_{m+1} u_{m+2} \cdots$. Hence $w \in u \mathbin{|||} v$.

Finally, consider the case that $u, v \in \Delta^\omega$. For every finite prefix $w' = u_1 v_1 u_2 v_2 \cdots u_m v_m \in \mathrm{pref}(w)$, for some $m \geq 1$, we know that $\rho_m = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell)$ is such that $\alpha_1, \beta_\ell \in \Delta^*$, $\alpha_j \in \Delta^+$, for all $1 < j \leq \ell$, and $\beta_j \in \Delta^+$, for all $1 \leq j < \ell$. We obtain that $\lim_{\ell \to \infty} \alpha_1 \beta_1 \alpha_2 \beta_2 \cdots \alpha_\ell \beta_\ell = \lim_{m \to \infty} u_1 v_1 u_2 v_2 \cdots u_m v_m = w$. Hence $w \in u \mathbin{|||} v$.

(2) By using Lemma 6.3.4(3) this follows from (1). □

Lemma 6.3.7 thus serves as an alternative definition of a shuffle of two (possibly infinite) words. With this alternative definition, commutativity of (fairly) shuffling two (possibly infinite) words follows immediately.

Theorem 6.3.8. *Let* $u, v \in \Delta^\infty$. *Then*

(1) $u \mathbin{|||} v = v \mathbin{|||} u$ *and*

(2) $u \mathbin{||} v = v \mathbin{||} u$.

Proof. (1) By symmetry it suffices to prove that $u \mathbin{|||} v \subseteq v \mathbin{|||} u$. Let $w \in u \mathbin{|||} v$. By Lemma 6.3.7(1), $w = u_1 v_1 u_2 v_2 \cdots$, with $u_i, v_i \in \Delta^*$ for all $i \geq 1$, $u = u_1 u_2 \cdots$, and $v = v_1 v_2 \cdots$. Clearly we can also write w as $v_0 u_1 v_1 u_2 v_2 \cdots$, with $v_0 = \lambda$. Lemma 6.3.7(1) then implies that $w \in v \mathbin{|||} u$.

(2) Analogous. □

This theorem implies that also the (fair) shuffle of two (infinitary) languages is commutative.

Theorem 6.3.9. *Let* $L_1, L_2 \subseteq \Delta^\infty$. *Then*

(1) $L_1 \mathbin{|||} L_2 = L_2 \mathbin{|||} L_1$ *and*

(2) $L_1 \mathbin{||} L_2 = L_2 \mathbin{||} L_1$.

Proof. (1) By symmetry it suffices to prove that $L_1 \mathbin{|||} L_2 \subseteq L_2 \mathbin{|||} L_1$. Let $w \in L_1 \mathbin{|||} L_2$. Then there exist a $u \in L_1$ and a $v \in L_2$ such that $w \in u \mathbin{|||} v$. By Theorem 6.3.8(1) it now follows that $w \in v \mathbin{|||} u$ and hence $w \in L_2 \mathbin{|||} L_1$.

(2) Analogous. □

Recall from Lemma 6.3.4(1) that in case of finite words there is no need to distinguish shuffles from fair shuffles. The following results also follow immediately from Definition 6.3.1.

Lemma 6.3.10. *Let* $u, v \in \Delta^*$ *and let* $w \in u \mathbin{||} v$. *Then*

(1) $alph(w) = alph(u) \cup alph(v)$ *and*

(2) $|w| = |u| + |v|$. □

Note that in case u or v (or both) are infinite words, then a word w from the shuffle $u \mathbin{||} v$ does not necessarily contain all letters that are contained in u and v, unless the shuffle is fair.

Lemma 6.3.10 immediately implies that the language formed by the shuffles of two finite words is finite.

Corollary 6.3.11. *Let $u, v \in \Delta^*$. Then*

$$\#(u \parallel v) \leq (\#(alph(u) \cup alph(v)))^{|u|+|v|} \text{ and hence } u \parallel v \text{ is finite.} \qquad \square$$

Next we wonder whether the language formed by the (fair) shuffles of two possibly infinite words can be finite. It turns out that this is the case. In fact, the series of results below leads to an exact formulation (cf. Theorem 6.3.26) of the conditions that guarantee this.

Lemma 6.3.12. *Let $u, v \in \Delta^\infty$ and let $z \in \Delta^*$. Then*

(1) $\{z\}(u \parallel\!\mid v) \subseteq zu \parallel\!\mid v$ and

(2) $\{z\}(u \parallel v) \subseteq zu \parallel v$.

Proof. (1) Let $w \in \{z\}(u \parallel\!\mid v)$. Then $w = zw'$ for some $w' \in u \parallel\!\mid v$. By Lemma 6.3.7(1), $w' = u_1v_1u_2v_2\cdots$ for some $u_i, v_i \in \Delta^*$ for all $i \geq 1$, $u = u_1u_2\cdots$, and $v = v_1v_2\cdots$. Thus $w = zw' = zu_1v_1u_2v_2\cdots$ with $zu_1u_2\cdots = zu$. Hence $w \in zu \parallel\!\mid v$.

(2) Analogous. $\square$

Lemma 6.3.13. *Let $u, v \in \Delta^\infty$ and let $a, b \in \Delta$. Then*

(1) $au \parallel\!\mid bv = \{a\}(u \parallel\!\mid bv) \cup \{b\}(au \parallel\!\mid v)$ and

(2) $au \parallel bv = \{a\}(u \parallel bv) \cup \{b\}(au \parallel v)$.

Proof. (1) From Lemma 6.3.12(1) it follows that $\{a\}(u \parallel\!\mid bv) \subseteq au \parallel\!\mid bv$ and by Theorem 6.3.8(1) also $\{b\}(au \parallel\!\mid v) = \{b\}(v \parallel\!\mid au) \subseteq bv \parallel\!\mid au = au \parallel\!\mid bv$. Thus we are left with proving the inclusions in the statement from left to right. Let $w \in au \parallel\!\mid bv$.

By Lemma 6.3.7(1), $w = u_1v_1u_2v_2\cdots$ for some $u_i, v_i \in \Delta^*$ for all $i \geq 1$, $u_1u_2\cdots = au$, and $v_1v_2\cdots = bv$. We can distinguish the following two cases.

First let $k \geq 1$ be such that $u_k = au'_k$ and for all $1 \leq j < k$, $u_j = v_j = \lambda$. In this case $w \in \{a\}(u \parallel\!\mid bv)$.

Secondly, let $k \geq 1$ be such that $u_k = \lambda$, $v_k = bv'_k$, and for all $1 \leq j < k$, $u_j = v_j = \lambda$. In this case $w \in \{b\}(au \parallel\!\mid v)$.

Hence we conclude that $w \in \{a\}(u \parallel\!\mid bv) \cup \{b\}(au \parallel\!\mid v)$.

(2) Analogous. $\square$

Lemma 6.3.14. *Let $u_1, v_1 \in \Delta^*$ and let $u_2, v_2 \in \Delta^\infty$. Then*

(1) $(u_1 \parallel v_1)(u_2 \parallel\!\mid v_2) \subseteq u_1u_2 \parallel\!\mid v_1v_2$ and

(2) $(u_1 \parallel v_1)(u_2 \parallel v_2) \subseteq u_1u_2 \parallel v_1v_2$.

Proof. (1) First assume that $u_1 = \lambda$. Then $u_1 \mid\mid v_1 = \{v_1\}$ by Lemma 6.3.6. Moreover, by the commutativity of $\mid\mid\mid$ and Lemma 6.3.12(1), we have that $\{v_1\}(u_2 \mid\mid\mid v_2) \subseteq u_2 \mid\mid\mid v_1 v_2$. The case that $v_1 = \lambda$ is symmetric.

Next we proceed by induction on $|u_1| + |v_1|$. The cases $|u_1| + |v_1| = 0$ and $|u_1| + |v_1| = 1$ have already been dealt with. Thus assume that $|u_1| + |v_1| \geq 2$ with $u_1 = au_1'$ and $v_1 = bv_1'$ for some $a, b \in \Delta$ and some $u_1', v_1' \in \Delta^*$. Then by Lemma 6.3.13(2), $u_1 \mid\mid v_1 = au_1' \mid\mid bv_1' = \{a\}(u_1' \mid\mid bv_1') \cup \{b\}(au_1' \mid\mid v_1')$. This yields $(u_1 \mid\mid v_1)(u_2 \mid\mid\mid v_2) = \{a\}(u_1' \mid\mid bv_1')(u_2 \mid\mid\mid v_2) \cup \{b\}\ (au_1' \mid\mid v_1')\ (u_2 \mid\mid\mid v_2) \subseteq \{a\}\ (u_1'u_2 \mid\mid\mid bv_1'v_2) \cup \{b\}\ (au_1'u_2 \mid\mid\mid v_1'v_2) \subseteq (au_1'u_2 \mid\mid\mid bv_1'v_2) \cup (au_1'u_2 \mid\mid\mid bv_1'v_2) = (u_1u_2 \mid\mid\mid v_1v_2)$ by applying the induction hypothesis and Lemma 6.3.13 twice.

(2) Analogous. □

In the following example we show that the inclusions of this lemma can be proper.

Example 6.3.15. Let $\Delta = \{a, b\}$. Let $u = v = ab \in \Delta^*$. Then $u \mid\mid v \supseteq (a \mid\mid a)(b \mid\mid b)$ by Lemma 6.3.14(2). Since $abab \in u \mid\mid v$ and $(a \mid\mid a)(b \mid\mid b) = a^2b^2$, this inclusion is proper. □

Lemma 6.3.14 has the following direct consequences.

Corollary 6.3.16. *Let $u = u_1u_2 \cdots u_n$ and $v = v_1v_2 \cdots v_n$ be such that $u_i, v_i \in \Delta^*$, with $1 \leq i < n$, and $u_n, v_n \in \Delta^\infty$. Then*

(1) $(u_1 \mid\mid v_1)(u_2 \mid\mid v_2) \cdots (u_{n-1} \mid\mid v_{n-1})(u_n \mid\mid\mid v_n) \subseteq u \mid\mid\mid v$ and

(2) $(u_1 \mid\mid v_1)(u_2 \mid\mid v_2) \cdots (u_n \mid\mid v_n) \subseteq u \mid\mid v$. □

Corollary 6.3.17. *Let $u, v \in \Delta^\infty$. Then*

$$pref(u) \mid\mid pref(v) \subseteq pref(u \mid\mid\mid v). \qquad \square$$

The statement of this corollary holds also the other way around. This will be stated below as part of a more general equality. First we lift the statement of this corollary to languages.

Corollary 6.3.18. *Let $K, L \subseteq \Delta^\infty$. Then*

$$pref(K) \mid\mid pref(L) \subseteq pref(K \mid\mid\mid L).$$

Proof. Let $x \in \mathrm{pref}(K) \mid\mid \mathrm{pref}(L)$. Then by definition there exist a $u \in K$ and a $v \in L$ such that $x \in \mathrm{pref}(u) \mid\mid \mathrm{pref}(v)$, which according to Corollary 6.3.17 implies that $x \in \mathrm{pref}(u \mid\mid\mid v)$. Consequently, by definition $x \in \mathrm{pref}(K \mid\mid\mid L)$. □

Consequently, we obtain the following result and its extension to languages.

Lemma 6.3.19. *Let $u, v \in \Delta^\infty$. Then*

(1) $\mathit{pref}(u \mathbin{|||} v) \subseteq \mathit{pref}(u) \mathbin{|||} \mathit{pref}(v)$ and

(2) $\mathit{pref}(u \mathbin{||} v) \subseteq \mathit{pref}(u) \mathbin{||} \mathit{pref}(v)$.

Proof. (1) Let $z \in \mathrm{pref}(u \mathbin{|||} v)$. Then there exist $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n$, and x such that $z = u_1 v_1 u_2 v_2 \cdots u_{n-1} v_{n-1} x$, where $u_1 \in \Delta^*$, $u_2, u_3, \ldots, u_{n-1}$, $v_1, v_2, \ldots, v_{n-1} \in \Delta^+$, and $x \in \Delta^*$ are such that $x \in \mathrm{pref}(u_n v_n)$, with $u_n, v_n \in \Delta^*$, $u_1 u_2 \cdots u_n \in \mathrm{pref}(u)$, and $v_1 v_2 \cdots v_n \in \mathrm{pref}(v)$. Hence $z \in \mathrm{pref}(u) \mathbin{|||} \mathrm{pref}(v)$.

(2) Analogous. □

Lemma 6.3.20. *Let $K, L \subseteq \Delta^\infty$. Then*

(1) $\mathit{pref}(K \mathbin{|||} L) \subseteq \mathit{pref}(K) \mathbin{|||} \mathit{pref}(L)$ and

(2) $\mathit{pref}(K \mathbin{||} L) \subseteq \mathit{pref}(K) \mathbin{||} \mathit{pref}(L)$.

Proof. (1) Let $x \in \mathrm{pref}(K \mathbin{|||} L)$. Then by definition there exist a $u \in K$ and a $v \in L$ such that $x \in \mathrm{pref}(u \mathbin{|||} v)$. Consequently, Lemma 6.3.19(1) implies that $x \in \mathrm{pref}(u) \mathbin{|||} \mathrm{pref}(v)$. Hence, by definition, $x \in \mathrm{pref}(K) \mathbin{|||} \mathrm{pref}(L)$.

(2) Analogous. □

Now we are ready to present the aforementioned equality and its extension to languages, including the converses of the statements of Corollaries 6.3.17 and 6.3.18.

Theorem 6.3.21. *Let $u, v \in \Delta^\infty$ and let $K, L \subseteq \Delta^\infty$. Then*

(1) $\mathit{pref}(u \mathbin{|||} v) = \mathit{pref}(u) \mathbin{|||} \mathit{pref}(v) = \mathit{pref}(u) \mathbin{||} \mathit{pref}(v) = \mathit{pref}(u \mathbin{||} v)$ and

(2) $\mathit{pref}(K \mathbin{|||} L) = \mathit{pref}(K) \mathbin{|||} \mathit{pref}(L) = \mathit{pref}(K) \mathbin{||} \mathit{pref}(L) = \mathit{pref}(K \mathbin{||} L)$.

Proof. (1) By Lemmata 6.3.19(1) and 6.3.3(2), Corollary 6.3.17, and Lemmata 6.3.3(2) and 6.3.19(2) we obtain $\mathrm{pref}(u \mathbin{|||} v) \subseteq \mathrm{pref}(u) \mathbin{|||} \mathrm{pref}(v) \subseteq \mathrm{pref}(u) \mathbin{||} \mathrm{pref}(v) \subseteq \mathrm{pref}(u \mathbin{|||} v) \subseteq \mathrm{pref}(u \mathbin{||} v) \subseteq \mathrm{pref}(u) \mathbin{||} \mathrm{pref}(v)$, which proves the statement.

(2) Analogous by Lemmata 6.3.20(1) and 6.3.3(2), Corollary 6.3.18, and Lemmata 6.3.3(2) and 6.3.20(2). □

We now continue our quest for a precise formulation of the conditions under which the language formed by the (fair) shuffles of two possibly infinite words can be finite.

We begin by isolating the case that u and v are words over the unary alphabet $\{a\}$. Recall from Lemma 6.3.10 that whenever $u = a^k$ and $v = a^\ell$, for some $k, \ell \in \mathbb{N}$, then $u \mid\mid v = \{a^{k+\ell}\}$. However, if $u = a^\omega$, then $u \mid\mid v = u \mid\mid\mid v = \{u\}$.

Lemma 6.3.22. *Let $w \in \Delta^*$, let $a \in \Delta$, and let $k \geq 0$. Then*

(1) $wa^\omega \mid\mid\mid a^k = (w \mid\mid\mid a^k)\{a^\omega\}$ and

(2) $wa^\omega \mid\mid a^k = (w \mid\mid a^k)\{a^\omega\}$.

Proof. First observe that $\{a^\omega\} = a^\omega \mid\mid\mid \lambda = a^\omega \mid\mid \lambda$. Then by Lemma 6.3.14 we have $(w \mid\mid\mid a^k)\{a^\omega\} = (w \mid\mid a^k)(a^\omega \mid\mid\mid \lambda) \subseteq wa^\omega \mid\mid\mid a^k$ and $(w \mid\mid a^k)\{a^\omega\} = (w \mid\mid a^k)(a^\omega \mid\mid \lambda) \subseteq wa^\omega \mid\mid a^k$. Hence we are done once we have proven that $wa^\omega \mid\mid a^k \subseteq (w \mid\mid\mid a^k)\{a^\omega\}$.

Let $x \in wa^\omega \mid\mid a^k$. This means that there exist $n \geq 1$, $v_1 \in \Delta^*$, $v_2, v_3, \ldots, v_n, u_1, u_2, \ldots, u_{n-1} \in \Delta^+$, and $u_n \in \Delta^\omega$ such that $v_1 v_2 \cdots v_n = a^\ell$ for some $\ell \leq k$, $u_1 u_2 \cdots u_n = wa^\omega$, and $x = v_1 u_1 v_2 u_2 \cdots v_n u_n$. Without loss of generality we may assume that $v_1 v_2 \cdots v_n = a^k$. This can be seen as follows. If $v_1 v_2 \cdots v_n = a^\ell$ and $\ell < k$, then since $u_n = w_2 a^\omega$ for some suffix w_2 of w we have $x = v_1 u_1 v_2 u_2 \cdots v_n w_2 a^{k-\ell} a^\omega$.
In case $w_2 \neq \lambda$ we have $x = v_1 u_1 v_2 u_2 \cdots v_n u'_n v_{n+1} u_{n+1}$ with $u'_n = w_2$, $v_{n+1} = a^{k-\ell}$, and $u_{n+1} = a^\omega$.
In case $w_2 = \lambda$ we have $x = v_1 u_1 v_2 u_2 \cdots u_{n-1} v'_n u_n$ with $v'_n = v_n a^{k-\ell}$.
Hence from here we assume that $x = v_1 u_1 v_2 u_2 \cdots v_n u_n$ with $u_1 u_2 \cdots u_n = wa^\omega$ and $v_1 v_2 \cdots v_n = a^k$.

Suppose that $u_1 u_2 \cdots u_{n-1} \in \mathrm{pref}(w)$. Then for some suffix w_2 of w we have $u_1 u_2 \cdots u_{n-1} w_2 = w$ and $u_n = w_2 a^\omega$. Consequently, we thus have $v_1 u_1 v_2 u_2 \cdots v_{n-1} u_{n-1} v_n w_2 \in a^k \mid\mid\mid w = w \mid\mid\mid a^k$ and thus $x \in (w \mid\mid\mid a^k)\{a^\omega\}$.

In the case that $u_1 u_2 \cdots u_{n-1} \notin \mathrm{pref}(w)$ we have $u_1 u_2 \cdots u_{n-1} \in w\{a\}^*$. Let $m = \min\{1 \leq j \leq n-1 \mid u_1 u_2 \cdots u_j \in w\{a\}^*\}$, where min applied to a set of positive integers selects the smallest number among this set of integers. Thus $u_m = u_{m,1} u_{m,2}$ with $u_1 u_2 \cdots u_{m-1} u_{m,1} = w$ and $u_{m,2} = \{a\}^*$. Hence with $v_1 v_2 \cdots v_m = a^\ell$ for some $\ell \leq k$ we have $u_{m,2} v_{m+1} u_{m+1} v_{m+2} \cdots u_{n-1} v_n = a^p$ for some $p \geq k - l$.

Now we have $x = v_1 u_1 v_2 u_2 \cdots v_m u_{m,1} a^p a^\omega = v_1 u_1 v_2 u_2 \cdots v_m u_{m,1} a^{k-\ell} a^\omega$ and thus $x \in (a^k \mid\mid\mid w)\{a^\omega\} = (w \mid\mid\mid a^k)\{a^\omega\}$. $\square$

Whenever two nonempty words yield only one word as their shuffle, then it must be the case that those words are words over the same unary alphabet.

Lemma 6.3.23. *Let $u, v \in \Delta^\infty$ be such that both $u \neq \lambda$ and $v \neq \lambda$. Then*

(1) if $u \mathbin{|||} v = \{w\}$, for some $w \in \Delta^\infty$, then $u, v \in \{a\}^\infty$, for some $a \in \Delta$, and

(2) if $u \mathbin{||} v = \{w\}$, for some $w \in \Delta^\infty$, then $u, v \in \{a\}^\infty$, for some $a \in \Delta$.

Proof. (1) We prove the statement by contradiction, i.e. we assume that $\mathrm{alph}(u) \cup \mathrm{alph}(v)$ contains at least two elements.

First consider the case that $\mathrm{alph}(u) \setminus \mathrm{alph}(v) \neq \varnothing$. Let $b \in \mathrm{alph}(u) \setminus \mathrm{alph}(v)$. Hence $u = u_1 b u_2$ where $u_1 \in (\Delta \setminus \{b\})^*$ and $u_2 \in \Delta^\infty$. Let $v = cz$ for some $c \in \Delta \setminus \{b\}$ and $z \in (\Delta \setminus \{b\})^\infty$. Consider $w_1 = u_1 bcy$ and $w_2 = u_1 cby$, where $y \in u_2 \mathbin{|||} z$. Since $u_1 bc \in u_1 b \mathbin{|||} c$, Lemma 6.3.14(1) implies that $w_1 \in u \mathbin{|||} v$. Similarly $w_2 \in u \mathbin{|||} v$ because $u_1 cb \in u_1 b \mathbin{|||} c$. However, $b \neq c$ and thus $w_1 \neq w_2$, a contradiction.

Next consider the case that $\mathrm{alph}(u) = \mathrm{alph}(v)$. Hence $u = u_1 ab u_2$ for some $a, b \in \Delta$, $a \neq b$, $u_1 \in \{a\}^*$, and $u_2 \in \Delta^\infty$. Let $v = cz$ for some $c \in \Delta$ and $z \in \Delta^\infty$. Consider $w_1 = u_1 abcy$ and $w_2 = cu_1 aby$, where $y \in u_2 \mathbin{|||} z$. As above both $w_1, w_2 \in u \mathbin{|||} v$ but $w_1 \neq w_2$, a contradiction.

Both cases thus lead to a contradiction and hence $\#(\mathrm{alph}(u) \cup \mathrm{alph}(v)) = 1$, i.e. $u, v \in \{a\}^\infty$ for some $a \in \Delta$.

(2) This follows from (1) and Lemma 6.3.3(1) combined with the fact that $u \mathbin{|||} v \neq \varnothing$. □

In fact, by Lemmata 6.3.6 and 6.3.23 it now follows that the (fair) shuffles of two words form a singleton language if and only if either one of those original words is empty, or both are words over the same unary alphabet.

Corollary 6.3.24. *Let $u, v \in \Delta^\infty$. Then*

(1) $u \mathbin{|||} v = \{w\}$, for some $w \in \Delta^\infty$, if and only if either $u = \lambda$, or $v = \lambda$, or $u, v \in \{a\}^\infty$, for some $a \in \Delta$, and

(2) $u \mathbin{||} v = \{w\}$, for some $w \in \Delta^\infty$, if and only if either $u = \lambda$, or $v = \lambda$, or $u, v \in \{a\}^\infty$, for some $a \in \Delta$. □

Next we state the conditions under which the (fair) shuffles of an infinite and a second (possibly infinite) word form a finite language.

Lemma 6.3.25. *Let $u \in \Delta^\omega$ and let $v \in \Delta^\infty \setminus \{\lambda\}$. Then*

(1) $u \mathbin{|||} v$ is finite if and only if either $u = wa^\omega$ and $v \in \{a\}^*$, or $u = v = a^\omega$, for some $w \in \Delta^*$ and $a \in \Delta$, and

(2) $u \mid\mid v$ is finite if and only if either $u = wa^\omega$ and $v \in \{a\}^*$, or $u = v = a^\omega$, for some $w \in \Delta^*$ and $a \in \Delta$.

Proof. (1) (If) Follows directly from Lemma 6.3.22(1).

(Only if) Let $u \mid\mid\mid v$ be a finite set and let $u = b_1 b_2 \cdots$ with $b_i \in \Delta$ for all $i \geq 1$. Suppose first that $\mathrm{alph}(v) \setminus \mathrm{alph}(u) \neq \varnothing$. Then $v = v_1 c v_2$ for some $v_1 \in \Delta^*$, $c \in \Delta \setminus \mathrm{alph}(u)$, and $v_2 \in \Delta^\infty$. Now set, for all $i \geq 0$, $W_i = v_1 b_1 b_2 \cdots b_i c (b_{i+1} b_{i+2} \cdots \mid\mid\mid v_2)$. Since $v_1 b_1 b_2 \cdots b_i c \in b_1 b_2 \cdots b_i \mid\mid\mid v_1 c$, Lemma 6.3.14(1) implies that $W_i \subseteq u \mid\mid\mid v$ for all $i \geq 0$. For each $i \geq 0$, all words in W_i have a c at position $|v_1| + i + 1$ and for all $k > i$, all words in W_k have b_i at position $|v_1| + i + 1$. Since $c \neq b_i$ for all $i \geq 1$, this implies that the W_i are mutually disjoint. Since they are not empty this implies that $\bigcup_{i \geq 0} W_i$ is infinite and hence $u \mid\mid\mid v$ is infinite, a contradiction.

Hence it must be the case that $\mathrm{alph}(v) \subseteq \mathrm{alph}(u)$. Now suppose that there exist $x \in \Delta^*$ and $y \in \Delta^\omega$ such that $u = xy$ and $\mathrm{alph}(v) \setminus \mathrm{alph}(y) \neq \varnothing$. Then by the same reasoning as given above we know that $y \mid\mid\mid v$ is infinite and since by Lemma 6.3.12(1) $x(y \mid\mid\mid v) \subseteq xy \mid\mid\mid v = u \mid\mid\mid v$ it follows that $u \mid\mid\mid v$ is infinite, again a contradiction.

Hence every symbol in v occurs infinitely often in u. Suppose that there are (at least) two different symbols occurring infinitely often in u. Thus for all $N \in \mathbb{N}$ there exists a $k_N \geq N$ such that $b_{k_N} \neq c$, where c is the first symbol of v. Thus we have $v = cv'$ with $c \in \Delta$ and $v' \in \Delta^\infty$. Let $u_N \in \Delta^\omega$ be such that $u = b_1 b_2 \cdots b_{k_N} u_N$. Set for all $N \geq 0$, $W_N = b_1 b_2 \cdots b_{k_N - 1} c b_{k_N} (u_N \mid\mid\mid v')$. Since $b_1 b_2 \cdots b_{k_N-1} c b_{k_N} \in b_1 b_2 \cdots b_{k_N - 1} b_{k_N} \mid\mid\mid c$, Lemma 6.3.14(1) implies that $W_N \subseteq u \mid\mid\mid v$ for all $N \geq 0$. For each $N \geq 0$, all words in W_N have c at position k_N and for all N' such that $k_{N'} > k_N$, all words in $W_{N'}$ have b_{k_N} at position k_N. Since $c \neq b_{k_N}$ this implies that $W_N \cap W_{N'} = \varnothing$ whenever $k_{N'} > k_N$. Since $(k_N, N \geq 0)$ contains an infinite strictly increasing subsequence $k_{N_1} > k_{N_2} > \cdots$ this implies that $\bigcup_{N \in \mathbb{N}} W_N$ is infinite and hence $u \mid\mid\mid v$ is infinite, a contradiction once again.

Thus it must be the case that at most one symbol occurs infinitely often in u. Combining this with the already established fact that every symbol in v occurs infinitely often in u, we obtain that $u = wa^\omega$ for some $w \in \Delta^*$, $a \in \Delta$ and $v \in \{a\}^\infty$.

Finally assume that $\mathrm{alph}(w) \setminus \{a\} \neq \varnothing$ and suppose that $v = a^\omega$. then $a^i w a^\omega \neq a^j w a^\omega$ if $i \neq j$, but $a^i w a^\omega \subseteq u \mid\mid\mid v$ for all $i \geq 0$. Thus also in this case $u \mid\mid\mid v$ is infinite, a contradiction. Hence if $v = a^\omega$, then $u = a^\omega$ and $u \mid\mid\mid v = \{a^\omega\}$. If $v \neq a^\omega$, then $v = a^k$ for some $k \geq 1$ and $u = wa^\omega$. In this case $u \mid\mid\mid v = (w \mid\mid\mid a^k)\{a^\omega\}$ by Lemma 6.3.22(1) and thus $u \mid\mid\mid v$ is finite.

(2) (If) Follows directly from Lemma 6.3.22(2).

(Only if) If $u \mid\mid v$ is a finite set, then by Lemma 6.3.3(1) also $u \mid\mid\mid v$ is a finite set and the statement follows from (1). $\square$

As a summary of the results obtained in Corollaries 6.3.11 and 6.3.24 and Lemma 6.3.25 we can now formulate the conditions under which the (fair) shuffles of two words form a finite language.

Theorem 6.3.26. *Let $u, v \in \Delta^\infty$. Then*

(1) $u \mid\mid\mid v$ is finite if and only if either $u, v \in \Delta^*$, or $u = \lambda$, or $v = \lambda$, or there exists an $a \in \Delta$ such that $u, v \in \{a\}^\infty$, or there exists a $w \in \Delta^*$ such that either $u = wa^\omega$ and $v \in \{a\}^*$, or $v = wa^\omega$ and $u \in \{a\}^*$, and

(2) $u \mid\mid v$ is finite if and only if either $u, v \in \Delta^*$, or $u = \lambda$, or $v = \lambda$, or there exists an $a \in \Delta$ such that $u, v \in \{a\}^\infty$, or there exists a $w \in \Delta^*$ such that either $u = wa^\omega$ and $v \in \{a\}^*$, or $v = wa^\omega$ and $u \in \{a\}^*$. $\square$

6.3.3 Commutativity and Associativity

For later use of shuffles in the context of team automata, it is important to know that shuffles are commutative and associative. In Subsection 6.3.2 we showed the commutativity of the (fair) shuffles in Theorems 6.3.8 and 6.3.9 via the alternative definition of (fair) shuffles presented in Lemma 6.3.7. Before we deal with associativity we first present two lemmata that together provide a result (cf. Theorem 6.3.29) that has Theorem 6.3.8(1) as a direct corollary. This result actually is yet another alternative definition for the fair shuffle of two (possibly infinite) words. It sheds light on the particular characteristics of fair shuffles and it plays an important role in the remainder of this section.

First we need some auxiliary definitions. Let Δ be an alphabet. For each $i \in \mathbb{N}$ and $a \in \Delta$ we let $[a, i]$ be a distinct symbol. Let $[\Delta, i] = \{[a, i] \mid a \in \Delta\}$. Thus for all $i, j \in \mathbb{N}$ such that $i \neq j$, $[\Delta, i]$ and $[\Delta, j]$ are disjoint. We moreover assume, for all $i \in \mathbb{N}$, that Δ and $[\Delta, i]$ are disjoint. Let $i \in \mathbb{N}$. We define the homomorphisms $\beta_i : \Delta^* \to [\Delta, i]^*$ and $\overline{\beta}_i : [\Delta, i]^* \to \Delta^*$ by $\beta_i(a) = [a, i]$ and $\overline{\beta}_i([a, i]) = a$, respectively. Note that β_i and $\overline{\beta}_i$ are bijections. Intuitively, β_i is used to uniquely label every symbol in a word before this word is used in a shuffle, after which $\overline{\beta}_i$ can be used to remove this label again.

In addition we define the following homomorphisms. Let $i \in \mathbb{N}$ and let $J \subseteq \mathbb{N}$ be such that $i \notin J$. The homomorphism $\varphi_{i,J} : (\bigcup\{[\Delta, j] \mid j \in \{i\} \cup J\})^* \to \Delta^*$ is defined by $\varphi_{i,J}([a, i]) = a$ and $\varphi_{i,J}([a, j]) = \lambda$, for all $j \in J$, whereas the homomorphism $\psi_J : (\bigcup\{[\Delta, j] \mid j \in J\})^* \to \Delta^*$ is defined by $\psi_J([a, j]) = a$, for all $j \in J$. Note that $\varphi_{i,\varnothing} = \overline{\beta}_i$ and $\psi_{\{j\}} = \overline{\beta}_j$. Intuitively,

$\varphi_{i,J}$ is used to remove the label i from every symbol in a word that is labeled by i and to erase every other symbol from that word, whereas ψ_J simply removes all labels in J from every symbol in a word that is labeled by such a label from J.

Lemma 6.3.27. *Let $u, v \in \Delta^\infty$. Then, for all $i, j \in \mathbb{N}$ such that $i \neq j$,*

$$u \mathbin{|||} v \subseteq \psi_{\{i,j\}}(\varphi^{-1}_{i,\{j\}}(u) \cap \varphi^{-1}_{j,\{i\}}(v)).$$

Proof. Without loss of generality we assume that $i = 1$ and $j = 2$. Moreover, we prove only the case that $u \in \Delta^*$ and $v \in \Delta^\infty$. The proofs of the other cases are analogous.

Let $w \in u \mathbin{|||} v$. Hence $w = u_1v_1u_2v_2 \cdots u_nv_n$ with $n \geq 1$, $u_1 \in \Delta^*$, $u_2, u_3, \ldots, u_n, v_1, v_2, \ldots, v_{n-1} \in \Delta^+$, $v_n \in \Delta^\omega$, $u = u_1u_2 \cdots u_n$, and $v = v_1v_2 \cdots v_n$. Now consider $\overline{w} = \beta_1(u_1)\beta_2(v_1)\beta_1(u_2)\beta_2(v_2) \cdots \beta_1(u_n)\beta_2(v_n)$. Recall from the definitions of β_1, β_2, $\varphi_{1,\{2\}}$, and $\varphi_{2,\{1\}}$ that for all $a \in \Delta$, $\varphi_{1,\{2\}}(\beta_1(a)) = a$ and $\varphi_{1,\{2\}}(\beta_2(a)) = \lambda$. Hence it follows immediately that $\varphi_{1,\{2\}}(\overline{w}) = u$. Likewise, $\varphi_{2,\{1\}}(\overline{w}) = v$. Hence $\overline{w} \in \varphi^{-1}_{1,\{2\}}(u) \cap \varphi^{-1}_{2,\{1\}}(v)$. From the definitions of β_1, β_2, and $\psi_{\{1,2\}}$ we recall that for all $a \in \Delta$, $\psi_{\{1,2\}}(\beta_1(a)) = a$ and $\psi_{\{1,2\}}(\beta_2(a)) = a$. This implies that $\psi_{\{1,2\}}(\overline{w}) = w$ and we are done. □

Lemma 6.3.28. *Let $u, v \in \Delta^\infty$. Then, for all $i, j \in \mathbb{N}$ such that $i \neq j$,*

$$\psi_{\{i,j\}}(\varphi^{-1}_{i,\{j\}}(u) \cap \varphi^{-1}_{j,\{i\}}(v)) \subseteq u \mathbin{|||} v.$$

Proof. Without loss of generality we again assume that $i = 1$ and $j = 2$. Furthermore we again proof only the case that $u \in \Delta^*$ and $v \in \Delta^\infty$. The proofs of the other cases are analogous.

Let $w \in \psi_{\{1,2\}}(\varphi^{-1}_{1,\{2\}}(u) \cap \varphi^{-1}_{2,\{1\}}(v))$ and let $\overline{w} \in \varphi^{-1}_{1,\{2\}}(u) \cap \varphi^{-1}_{2,\{1\}}(v)$ be such that $\psi_{\{1,2\}}(\overline{w}) = w$. Since $\varphi_{1,\{2\}}(\overline{w}) = u$ there exist $m \geq 0$, $x_1, x_2, \ldots, x_m \in \Delta^*$, $x_{m+1} \in \Delta^\infty$, and $u_1, u_2, \ldots, u_m \in \Delta^+$ such that $\overline{w} = \beta_2(x_1)\beta_1(u_1)\beta_2(x_2)\beta_1(u_2) \cdots \beta_2(x_m)\beta_1(u_m)\beta_2(x_{m+1})$ and $u = u_1u_2 \cdots u_m$. Observe that the situation that $m = 0$ corresponds to the case that $u = \lambda$. Similarly, $\varphi_{2,\{1\}}(\overline{w}) = v$ and the fact that $v \neq \lambda$ imply that there exist $n \geq 1$, $y_1, y_2, \ldots, y_n \in \Delta^*$, $v_1, v_2, \ldots, v_{n-1} \in \Delta^+$, and $v_n \in \Delta^\omega$ such that $\overline{w} = \beta_1(y_1)\beta_2(v_1)\beta_1(y_2)\beta_2(v_2) \cdots \beta_1(y_n)\beta_2(v_n)$ and $v = v_1v_2 \cdots v_n$. We thus have the situation that $\beta_2(x_1)\beta_1(u_1)\beta_2(x_2)\beta_1(u_2) \cdots \beta_2(x_m)\beta_1(u_m)\beta_2(x_{m+1}) = \beta_1(y_1)\beta_2(v_1)\beta_1(y_2)\beta_2(v_2) \cdots \beta_1(y_n)\beta_2(v_n)$. Since $[\Delta, 1] \cap [\Delta, 2] = \varnothing$ it must be the case that either $\beta_2(x_1) = \lambda$ or $\beta_1(y_1) = \lambda$.

First assume that $\beta_2(x_1) = \lambda$, i.e. $x_1 = \lambda$. Now $v \in \Delta^\omega$ implies that $m \neq 0$. Thus we have that $\beta_1(u_1)\beta_2(x_2)\beta_1(u_2)\beta_2(x_3) \cdots \beta_2(x_m)\beta_1(u_m)\beta_2(x_{m+1}) =$

$\beta_1(y_1)\beta_2(v_1)\beta_1(y_2)\beta_2(v_2)\cdots\beta_1(y_n)\beta_2(v_n)$. Again by $[\Delta,1]\cap[\Delta,2]=\varnothing$ and from the fact that $u_i\in\Delta^+$ for all $1\le i\le m$, $v_j\in\Delta^+$ for all $1\le j\le n-1$, and $v_n\in\Delta^\omega$, we know that $m=n$ and, for all $1\le i\le n$, $\beta_1(u_i)=\beta_2(y_i)$ and $\beta_2(v_i)=\beta_2(x_{i+1})$. Consequently $w=\psi_{\{1,2\}}(\overline{w})=u_1v_1u_2v_2\cdots u_nv_n\in u\ |||\ v$.

Next assume that $\beta_1(y_1)=\lambda$, i.e. $y_1=\lambda$. In this case we thus have the situation that $\beta_2(x_1)\beta_1(u_1)\beta_2(x_2)\beta_1(u_2)\cdots\beta_2(x_m)\beta_1(u_m)\beta_2(x_{m+1})=\beta_2(v_1)\beta_1(y_2)\beta_2(v_2)\beta_1(y_3)\cdots\beta_1(y_n)\beta_2(v_n)$. Again by $[\Delta,1]\cap[\Delta,2]=\varnothing$ and from the fact that $u_i\in\Delta^+$ for all $1\le i\le m$, $v_j\in\Delta^+$ for all $1\le j\le n-1$, and $v_n\in\Delta^\omega$, we know that $n=m+1$, $\beta_1(u_i)=\beta_1(y_{i+1})$ and $\beta_2(v_i)=\beta_2(x_i)$, for all $1\le i\le m$, and $\beta_2(v_{m+1})=\beta_2(x_{m+1})$. Consequently $w=\psi_{\{1,2\}}(\overline{w})=v_1u_1v_2u_2\cdots v_mu_mv_{m+1}\in u\ |||\ v$. □

We now combine the two directly preceding lemmata to indeed obtain yet another alternative definition of the fair shuffle of two (possibly infinite) words. Note that since these lemmata use inverse homomorphisms based on the complete two words being shuffled. It therefore serves only as an alternative definition of the *fair* shuffle of these two words.

Theorem 6.3.29. *Let $u,v\in\Delta^\infty$. Then, for all $i,j\in\mathbb{N}$ such that $i\neq j$,*

$$u\ |||\ v=\psi_{\{i,j\}}(\varphi^{-1}_{i,\{j\}}(u)\cap\varphi^{-1}_{j,\{i\}}(v)).$$ □

This theorem now provides a different — rather elegant — proof of Theorem 6.3.8(1) since we know that intersection is commutative and thus $u\ |||\ v=\psi_{\{i,j\}}(\varphi^{-1}_{i,\{j\}}(u)\cap\varphi^{-1}_{j,\{i\}}(v))=\psi_{\{i,j\}}(\varphi^{-1}_{j,\{i\}}(v)\cap\varphi^{-1}_{i,\{j\}}(u))=v\ |||\ u$. The fair shuffle of two words can thus be obtained by applying a combination of (inverse) homomorphisms and intersection to those two words.

Example 6.3.30. (Example 6.3.2 continued) Note that we have $\varphi^{-1}_{1,\{2\}}(u)=\{\beta_2(x_1)\beta_1(a)\beta_2(x_2)\beta_1(b)\beta_2(x_3)\beta_1(c)\beta_2(x_4)\mid x_i\in\Delta^*,\ i\in[3],\ x_4\in\Delta^\infty\}=\{\beta_2(x_1)[a,1]\beta_2(x_2)[b,1]\beta_2(x_3)[c,1]\beta_2(x_4)\mid x_i\in\Delta^*,\ i\in[3],\ x_4\in\Delta^\infty\}$ and $\varphi^{-1}_{2,\{1\}}(v)=\{\beta_1(y_1)\beta_2(c)\beta_1(y_2)\beta_2(d)\beta_1(y_3)\mid y_i\in\Delta^*,\ i\in[2],\ y_3\in\Delta^\infty\}=\{\beta_1(y_1)[c,2]\beta_1(y_2)[d,2]\beta_1(y_3)\mid y_i\in\Delta^*,\ i\in[2],\ y_3\in\Delta^\infty\}$. Thus, e.g., $[a,1][c,2][b,1][d,2][c,1]\in\varphi^{-1}_{1,\{2\}}(u)\cap\varphi^{-1}_{2,\{1\}}(v)$ and hence we now obtain that $\psi_{\{1,2\}}([a,1][c,2][b,1][d,2][c,1])=acbdc\in\psi_{\{1,2\}}(\varphi^{-1}_{1,\{2\}}(u)\cap\varphi^{-1}_{2,\{1\}}(v))$. Finally, note that in Example 6.3.2 we have seen that indeed $acbdc\in u\ |||\ v$. □

This example shows why the construction $\psi_{\{i,j\}}(\varphi^{-1}_{i,\{j\}}(u)\cap\varphi^{-1}_{j,\{i\}}(v))$, with $u,v\in\Delta^\infty$ and $i\neq j\in\mathbb{N}$, in general does not equal $u\ ||\ v$: the inverse homomorphisms are "fair" in the sense that they take only complete words as input.

It remains to prove that (fairly) shuffling is associative. The remainder of this subsection is devoted to this. The setup is as follows. We first use Theorem 6.3.29 to prove the associativity of fairly shuffling (cf. Theorem 6.3.32). Lemma 6.3.4(1) then implies that associativity remains to be proven only in case infinite words are involved. To this aim we subsequently relate the shuffles of possibly infinite words to the shuffles of the finite prefixes of those possibly infinite words (cf. Theorem 6.3.49). We then conclude by using this result to prove associativity (cf. Theorem 6.3.51).

The following lemma streamlines the proof of the result succeeding it, which states that fairly shuffling is associative.

Lemma 6.3.31. *Let $u, v, w \in \Delta^\infty$. Let $i_1, i_2, i_3 \in \mathbb{N}$ be three different integers and let $j \in \mathbb{N}$ be different from i_1. Then*

$$\psi_{\{i_1,j\}}(\varphi^{-1}_{i_1,\{j\}}(u) \cap \varphi^{-1}_{j,\{i_1\}}(\psi_{\{i_2,i_3\}}(\varphi^{-1}_{i_2,\{i_3\}}(v) \cap \varphi^{-1}_{i_3,\{i_2\}}(w)))) =$$
$$\psi_{\{i_1,i_2,i_3\}}(\varphi^{-1}_{i_1,\{i_2,i_3\}}(u) \cap \varphi^{-1}_{i_2,\{i_1,i_3\}}(v) \cap \varphi^{-1}_{i_3,\{i_1,i_2\}}(w)).$$

Proof. Without loss of generality we assume that $i_1 = 1$, $i_2 = 2$, $i_3 = 3$, and $j \neq 1$.

($\subseteq$) Let $z \in \psi_{\{j,1\}}(\varphi^{-1}_{1,\{j\}}(u) \cap \varphi^{-1}_{j,\{1\}}(\psi_{\{2,3\}}(\varphi^{-1}_{2,\{3\}}(v) \cap \varphi^{-1}_{3,\{2\}}(w))))$ and let $\overline{z} \in \varphi^{-1}_{1,\{j\}}(u) \cap \varphi^{-1}_{j,\{1\}}(\psi_{\{2,3\}}(\varphi^{-1}_{2,\{3\}}(v) \cap \varphi^{-1}_{3,\{2\}}(w)))$ be such that $\psi_{\{j,1\}}(\overline{z}) = z$. Let $x \in \psi_{\{2,3\}}(\varphi^{-1}_{2,\{3\}}(v) \cap \varphi^{-1}_{3,\{2\}}(w))$ be such that $\overline{z} \in \varphi^{-1}_{1,\{j\}}(u) \cap \varphi^{-1}_{j,\{1\}}(x)$. Let $\overline{x} \in \varphi^{-1}_{2,\{3\}}(v) \cap \varphi^{-1}_{3,\{2\}}(w)$ be such that $\psi_{\{2,3\}}(\overline{x}) = x$. Hence $\overline{x}$ is of the form $\overline{x} = b_1c_1b_2c_2\cdots$ such that for all $i \geq 1$, $b_i \in [\Delta, 2] \cup \{\lambda\}$ and $c_i \in [\Delta, 3] \cup \{\lambda\}$, $\overline{\beta}_2(b_1b_2\cdots) = v$, and $\overline{\beta}_3(c_1c_2\cdots) = w$. Furthermore $\overline{z}$ is of the form $\overline{z} = a_1\overline{b}_1\overline{c}_1a_2\overline{b}_2\overline{c}_2\cdots$ such that for all $i \geq 1$, $a_i \in [\Delta, 1] \cup \{\lambda\}$ and $\overline{b}_i, \overline{c}_i \in [\Delta, j] \cup \{\lambda\}$, $\overline{\beta}_1(a_1a_2\cdots) = u$, and $\overline{\beta}_j(\overline{b}_1\overline{c}_1\overline{b}_2\overline{c}_2\cdots) = \psi_{\{2,3\}}(b_1c_1b_2c_2\cdots)$ is such that $\overline{\beta}_j(\overline{b}_1\overline{b}_2\cdots) = \overline{\beta}_2(b_1b_2\cdots) = v$ and $\overline{\beta}_j(\overline{c}_1\overline{c}_2\cdots) = \overline{\beta}_3(c_1c_2\cdots) = w$. Now consider $\overline{\overline{z}} = a_1\beta_2(\overline{\beta}_j(\overline{b}_1))\beta_3(\overline{\beta}_j(\overline{c}_1))a_2\beta_2(\overline{\beta}_j(\overline{b}_2))\beta_3(\overline{\beta}_j(\overline{c}_2))\cdots$. Since $\overline{\beta}_1(a_1a_2\cdots) = u$, $\overline{\beta}_2(\beta_2(\overline{\beta}_j(\overline{b}_1))\beta_2(\overline{\beta}_j(\overline{b}_2))\cdots) = \overline{\beta}_j(\overline{b}_1\overline{b}_2\cdots) = v$, and $\overline{\beta}_3(\beta_3(\overline{\beta}_j(\overline{c}_1))\beta_3(\overline{\beta}_j(\overline{c}_2))\cdots) = \overline{\beta}_j(\overline{c}_1\overline{c}_2\cdots) = w$, we know that $\varphi_{1,\{2,3\}}(\overline{\overline{z}}) = u$, $\varphi_{2,\{1,3\}}(\overline{\overline{z}}) = v$, and $\varphi_{3,\{1,2\}}(\overline{\overline{z}}) = w$. Hence $\overline{\overline{z}} \in \varphi^{-1}_{1,\{2,3\}}(u) \cap \varphi^{-1}_{2,\{1,3\}}(v) \cap \varphi^{-1}_{3,\{1,2\}}(w)$ and $\psi_{\{1,2,3\}}(\overline{\overline{z}}) = \psi_{\{j,1\}}(\overline{z}) = z$.

($\supseteq$) Let $z \in \psi_{\{1,2,3\}}(\varphi^{-1}_{1,\{2,3\}}(u) \cap \varphi^{-1}_{2,\{1,3\}}(v) \cap \varphi^{-1}_{3,\{1,2\}}(w))$ and let $\overline{z} \in \varphi^{-1}_{1,\{2,3\}}(u) \cap \varphi^{-1}_{2,\{1,3\}}(v) \cap \varphi^{-1}_{3,\{1,2\}}(w)$ be such that $\psi_{\{1,2,3\}}(\overline{z}) = z$. Hence $\overline{z}$ is of the form $\overline{z} = a_1b_1c_1a_2b_2c_2\cdots$ such that for all $i \geq 1$, $a_i \in [\Delta, 1] \cup \{\lambda\}$, $b_i \in [\Delta, 2] \cup \{\lambda\}$, and $c_i \in [\Delta, 3] \cup \{\lambda\}$, $\overline{\beta}_1(a_1a_2\cdots) = u$, $\overline{\beta}_2(b_1b_2\cdots) = v$, and $\overline{\beta}_3(c_1c_2\cdots) = w$. Let $\overline{u} = a_1\alpha_1a_2\alpha_2\cdots$, with $\alpha_i \in ([\Delta, j] \cup \{\lambda\})^*$, be such that for all $i \geq 1$, $\overline{\beta}_j(\alpha_i) = \psi_{\{2,3\}}(b_ic_i)$. Then clearly $\overline{u} \in \varphi^{-1}_{1,\{j\}}(u)$. Next let $\overline{x} = b_1c_1b_2c_2\cdots$. Then $\overline{x} \in \varphi^{-1}_{2,\{3\}}(v) \cap \varphi^{-1}_{3,\{2\}}(w)$. Since for all $i \geq 1$, $\varphi_{j,\{1\}}(\alpha_i) = \overline{\beta}_j(\alpha_i) = \psi_{\{2,3\}}(b_ic_i)$ and $a_i \in [\Delta, 1] \cup \{\lambda\}$, it follows that

$\overline{u} \in \varphi^{-1}_{j,\{1\}}(\psi_{\{2,3\}}(\overline{x}))$. Thus $\overline{u} \in \varphi^{-1}_{1,\{j\}}(u) \cap \varphi^{-1}_{j,\{1\}}(\psi_{\{2,3\}}(\overline{x}))$. Finally, the fact that for all $i \geq 1$, $\overline{\beta}_j(\alpha_i) = \psi_{\{2,3\}}(b_i c_i)$ now implies that $\psi_{\{j,1\}}(\overline{u}) = \psi_{\{1,2,3\}}(\overline{z}) = z$. $\square$

Theorem 6.3.32. *Let $u, v, w \in \Delta^\infty$ and let $L_1, L_2, L_3 \subseteq \Delta^\infty$. Then*

(1) $\{u\} \mathbin{|||} (v \mathbin{|||} w) = (u \mathbin{|||} v) \mathbin{|||} \{w\}$ and

(2) $L_1 \mathbin{|||} (L_2 \mathbin{|||} L_3) = (L_1 \mathbin{|||} L_2) \mathbin{|||} L_3$.

Proof. (1) From Definition 6.3.1, Theorem 6.3.29, and Lemma 6.3.31 we obtain that $\{u\} \mathbin{|||} (v \mathbin{|||} w) = \{x \mid \exists y \in v \mathbin{|||} w : \ x \in u \mathbin{|||} y\} = \{x \mid \exists y \in \psi_{\{k,\ell\}}(\varphi^{-1}_{k,\{\ell\}}(v) \cap \varphi^{-1}_{\ell,\{k\}}(w)) : \ x \in \psi_{\{i,j\}}(\varphi^{-1}_{i,\{j\}}(u) \cap \varphi^{-1}_{j,\{i\}}(y)),\ i,j,k,\ell \in \mathbb{N},\ i \neq j,\ k \neq \ell\} = \{x \mid x \in \psi_{\{i,j\}}(\varphi^{-1}_{i,\{j\}}(u) \cap \varphi^{-1}_{j,\{i\}}(\psi_{\{k,\ell\}}(\varphi^{-1}_{k,\{\ell\}}(u) \cap \varphi^{-1}_{\ell,\{k\}}(v)))),\ i,j,k,\ell \in \mathbb{N},\ i \neq j,\ k \neq \ell\} = \{x \mid x \in \psi_{\{i,k,\ell\}}(\varphi^{-1}_{i,\{k,\ell\}}(u) \cap \varphi^{-1}_{k,\{i,\ell\}}(v) \cap \varphi^{-1}_{\ell,\{i,k\}}(w)),\ i,k,\ell \in \mathbb{N},\ i \neq k,\ k \neq \ell,\ \ell \neq i\} = \{x \mid x \in \psi_{\{j,\ell\}}(\varphi^{-1}_{j,\{\ell\}}(\psi_{\{i,k\}}(\varphi^{-1}_{i,\{k\}}(u) \cap \varphi^{-1}_{k,\{i\}}(v))) \cap \varphi^{-1}_{\ell,\{j\}}(w),\ i,j,k,\ell \in \mathbb{N},\ i \neq k,\ j \neq \ell\} = \{x \mid \exists z \in \psi_{\{i,k\}}(\varphi^{-1}_{i,\{k\}}(u) \cap \varphi^{-1}_{k,\{i\}}(v)) : \ x \in \psi_{\{j,\ell\}}(\varphi^{-1}_{j,\{\ell\}}(z) \cap \varphi^{-1}_{\ell,\{j\}}(w),\ i,j,k,\ell \in \mathbb{N},\ i \neq k,\ j \neq \ell\} = \{x \mid \exists z \in u \mathbin{|||} v : \ z \in z \mathbin{|||} w\} = (u \mathbin{|||} v) \mathbin{|||} \{w\}$.

(2) By definition and (1) we obtain $L_1 \mathbin{|||} (L_2 \mathbin{|||} L_3) = \{x \in u \mathbin{|||} y \mid u \in L_1,\ y \in L_2 \mathbin{|||} L_3\} = \{x \in \{u\} \mathbin{|||} (v \mathbin{|||} w) \mid u \in L_1,\ v \in L_2,\ w \in L_3\} = \{x \in (u \mathbin{|||} v) \mathbin{|||} \{w\} \mid u \in L_1,\ v \in L_2,\ w \in L_3\} = \{x \in z \mathbin{|||} w \mid z \in L_1 \mathbin{|||} L_2,\ w \in L_3\} = (L_1 \mathbin{|||} L_2) \mathbin{|||} L_3$. $\square$

Due to Lemma 6.3.4(1) this result implies that also in the special case that we deal with finite words (finitary languages) only, shuffling is associative.

Corollary 6.3.33. *Let $u, v, w \in \Delta^*$ and let $L_1, L_2, L_3 \subseteq \Delta^*$. Then*

(1) $\{u\} \mathbin{||} (v \mathbin{||} w) = (u \mathbin{||} v) \mathbin{||} \{w\}$ and

(2) $L_1 \mathbin{||} (L_2 \mathbin{||} L_3) = (L_1 \mathbin{||} L_2) \mathbin{||} L_3$. $\square$

Hence what remains is the case that infinite words are involved. To this aim we now seek to express the shuffles of possibly infinite words in terms of shuffles of their finite prefixes, which obviously are fair shuffles.

We begin by defining (u, v)-*decompositions* as a way to interleave the finite words u and v by alternating sequences from u and v. The construction of these (u, v)-decompositions resembles a construction used in the proof of Lemma 6.3.7.

Definition 6.3.34. *Let $w \in \Delta^*$. Then* *a* decomposition *of w is a sequence $d = (u_1, v_1, u_2, v_2, \ldots, u_n, v_n)$ such that $n \geq 1$, $u_1 \in \Delta^*$, $u_2, u_3, \ldots, u_n, v_1, v_2, \ldots, v_{n-1} \in \Delta^+$, $v_n \in \Delta^*$, and $w = u_1 v_1 u_2 v_2 \cdots u_n v_n$.*

*If $u_1 u_2 \cdots u_n = u$ and $v_1 v_2 \cdots v_n = v$, then d is also called a (u, v)-*decomposition *of w.* □

Together with Definition 6.3.1(1) this leads to the following result.

Lemma 6.3.35. *Let $u, v, w \in \Delta^*$. Then*

there exists a (u, v)-decomposition of w if and only if $w \in u \mid\mid v$. □

Note that a shuffle $w \in u \mid\mid v$ may have several decompositions.

Example 6.3.36. Let $\Delta = \{a, b, c\}$. Let $u, v \in \Delta^*$ be such that $u = aba$ and $v = babc$. Clearly $w = abababc \in u \mid\mid v$. Note that both $d_1 = (a, ba, ba, bc)$ and $d_2 = (aba, babc)$ are (u, v)-decompositions of w. Hence w does not have a unique decomposition.

Note that also $w' = babcaba \in u \mid\mid v$. It is however easy to see that in this case $(\lambda, babc, aba, \lambda)$ is the unique (u, v)-decomposition of w'. □

If $d = (u_1, v_1, u_2, v_2, \ldots, u_n v_n)$ is a (u, v)-decomposition of a word z, then n intuitively is the number of alternations of sequences from u and v that form $z = u_1 v_1 u_2 v_2 \cdots u_n v_n$.

Definition 6.3.37. *Let $d = (u_1, v_1, u_2, v_2, \ldots, u_n, v_n)$, for some $n \in \mathbb{N}$, be a (u, v)-decomposition. Then*

n is the norm *of d, denoted by $\mid\mid d \mid\mid$.* □

Definition 6.3.38. *Let $d = (x_1, y_1, x_2, y_2, \ldots, x_k, y_k)$, for some $k \in \mathbb{N}$, and $d' = (u_1, v_1, u_2, v_2, \ldots, u_n, v_n)$, for some $n \in \mathbb{N}$, be two decompositions of two words over an alphabet Δ. Then*

(1) d directly precedes *d' if $k \leq n$ and for all $1 \leq j \leq k-1$, $x_j = u_j$ and $y_j = v_j$, and, moreover, one of the following three cases holds. Either*

(a) $k = n$, $x_k = u_k$, and $y_k a = v_k$, for some $a \in \Delta$, or

(b) $k = n$, $y_k = v_k = \lambda$, and $x_k a = u_k$, for some $a \in \Delta$, or

(c) $k = n - 1$, $y_k \neq \lambda$, $v_{k+1} = \lambda$, and $u_{k+1} = a$, for some $a \in \Delta$, and

(2) d precedes *d' if there exist decompositions $d_0, d_1, \ldots, d_\ell$ such that $\ell \geq 0$, $d = d_0$, $d' = d_\ell$, and for all $0 \leq j \leq \ell - 1$, d_j directly precedes d_{j+1}.* □

Note that if d and d' are two decompositions such that d directly precedes d', then $\|\,d'\| = \|\,d\,\|$ or $\|\,d'\| = \|\,d\,\| + 1$. Hence if d precedes d', then $\|\,d'\| \geq \|\,d\,\|$.

It is not difficult to see that whenever a decomposition d precedes a decomposition d', then d decomposes a prefix of the word that d' decomposes. In fact, we have the following result.

Lemma 6.3.39. *Let $d = (x_1, y_1, x_2, y_2, \ldots, x_k, y_k)$, for some $k \in \mathbb{N}$, and $d' = (u_1, v_1, u_2, v_2, \ldots, u_n, v_n)$, for some $n \in \mathbb{N}$, be two decompositions — of two words over an alphabet Δ — such that d precedes d'. Then*

$x_1x_2\cdots x_k \in \mathit{pref}(u_1u_2\cdots u_n)$, $y_1y_2\cdots y_k \in \mathit{pref}(v_1v_2\cdots v_n)$, and $x_1y_1x_2y_2\cdots x_ky_k \in \mathit{pref}(u_1v_1u_2v_2\cdots u_nv_n)$.

Proof. If $d = d'$ there is nothing to prove, so let us assume that $d \neq d'$. From Definition 6.3.38 it is clear that the statement holds in case d immediately precedes d'.

If d precedes d', then there exist (s_j, t_j)-decompositions d_j of words $w_j \in \Delta^*$ with $0 \leq j \leq \ell$, for some $\ell \geq 1$, such that $d_0 = d$, $d_\ell = d'$, and d_j immediately precedes d_{j+1}, for all $0 \leq j < \ell$. Thus, for all $0 \leq j < \ell - 1$, $s_j \in \mathrm{pref}(s_{j+1})$, $t_j \in \mathrm{pref}(t_{j+1})$, and $w_j \in \mathrm{pref}(w_{j+1})$. Hence $s_0 = x_1x_2\cdots x_k \in \mathrm{pref}(s_\ell) = \mathrm{pref}(u_1u_2\cdots u_n)$, $t_0 = y_1y_2\cdots y_k \in \mathrm{pref}(t_\ell) = \mathrm{pref}(v_1v_2\cdots v_n)$, and $w_0 = x_1y_1x_2y_2\cdots x_ky_k \in \mathrm{pref}(w_\ell) = \mathrm{pref}(u_1v_1u_2v_2\cdots u_nv_n)$. □

A sequence of decompositions — of words w_i into words u_i and words v_i, with $i \geq 0$ — preceding each other, uniquely defines the limit of the words w_i as an element of the shuffle of the limits of the words u_i and the words v_i.

Lemma 6.3.40. *For all $i \geq 0$, let d_i be a (u_i, v_i)-decomposition — of a word w_i over Δ — such that d_i precedes d_{i+1}. Then*

$u = \lim_{i\to\infty} u_i$, $v = \lim_{i\to\infty} v_i$, and $w = \lim_{i\to\infty} w_i$ exist, and $w \in u \,\|\, v$.

Proof. By Lemma 6.3.39 it follows that $u_i \leq u_{i+1}$, $v_i \leq v_{i+1}$, and $w_i \leq w_{i+1}$, for all $i \geq 0$, so indeed u, v, and w exist and we only have to prove that $w \in u \,\|\, v$. We distinguish two cases.

First we consider the case that there exists an $N \in \mathbb{N}$ such that $\|\,d_i\| = \|\,d_N\|$ for all $i \geq N$. Let $N_0 \in \mathbb{N}$ be such an N. Again we distinguish two cases.

Let us assume first that, for all $i \geq N_0$, if $d_i = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$, then $y_n = \lambda$. Consequently, for all $i \geq N_0$, $v_i = v_{N_0}$. From $u_i \leq u_{i+1}$, for all $i \geq 0$, we infer that for all $i > N_0$ there exist $z_i \in \Delta^*$ such that $u_{i+1} = u_iz_i$. Observe that $u = \lim_{i\to\infty} u_i = u_{N_0} \lim_{i\to\infty} z_1z_2\cdots z_{i-N_0}$. Thus

we obtain that for all $i > N_0$ we have $w_i = w_{N_0} z_1 z_2 \cdots z_{i-N_0}$. Since $w_{N_0} \in u_{N_0} \mid\mid v_{N_0}$ by Lemma 6.3.35, we conclude that $w = \lim_{i\to\infty} w_i \in (u_{N_0} \mid\mid v_{N_0}) \lim_{i\to\infty} z_1 z_2 \cdots z_{i-N_0} = (u_{N_0} \mid\mid v_{N_0})\ (\lim_{i\to\infty} z_1 z_2 \cdots z_{i-N_0} \mid\mid \lambda) \subseteq u \mid\mid v_{N_0} \subseteq u \mid\mid v$ by Lemma 6.3.14(2) and the definition of u.
Next assume there exist an $i \geq N_0$ such that $d_i = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$ with $y_n \neq \lambda$. Let ℓ_0 be the smallest such i. Thus, for all $i \geq \ell_0$, $u_i = u_{\ell_0}$. From $v_i \leq v_{i+1}$, for all $i \geq 0$, we infer that for all $i > \ell_0$ there exist $z_i \in \Delta^*$ such that $v_{i+1} = v_i z_i$. Observe that $v = \lim_{i\to\infty} v_i = v_{\ell_0} \lim_{i\to\infty} z_1 z_2 \cdots z_{i-v_0}$. Thus for all $i > \ell_0$ we have $w_i = w_{\ell_0} z_1 z_2 \cdots z_{i-\ell_0}$. Since $w_{\ell_0} \in u_{\ell_0} \mid\mid v_{\ell_0}$ by Lemma 6.3.35, we conclude that $w = \lim_{i\to\infty} w_i \in (u_{\ell_0} \mid\mid v_{\ell_0}) \lim_{i\to\infty} z_1 z_2 \cdots z_{i-\ell_0} = (u_{\ell_0} \mid\mid v_{\ell_0})(\lambda \mid\mid \lim_{i\to\infty} z_1 z_2 \cdots z_{i-\ell_0}) \subseteq u_{\ell_0} \mid\mid v \subseteq u \mid\mid v$ by Lemma 6.3.14(2) and the definition of u.

Now we move to the case that for all $N \in \mathbb{N}$ there exists a $k \in \mathbb{N}$ such that $\mid\mid d_k \mid\mid \geq N$. Let $j_1, j_2, \ldots \in \mathbb{N}$ be the (unique) infinite sequence of integers such that for all $i \in \mathbb{N}$, $\mid\mid d_{j_i} \mid\mid < \mid\mid d_{j_{i+1}} \mid\mid$ and $\mid\mid d_\ell \mid\mid = \mid\mid d_{j_i} \mid\mid$ for all $j_i \leq \ell < j_{i+1}$. Since $\mid\mid d_0 \mid\mid \leq \mid\mid d_1 \mid\mid \leq \cdots$ is an unbounded sequence of integers we know that the j_i as just described exist. Since each d_{j_i} precedes $d_{j_{i+1}}$, Definition 6.3.38 implies that there exist $x_1, x_2, \ldots, y_1, y_2, \ldots, s_1, s_2, \ldots, t_1, t_2, \cdots \in \Delta^*$ such that $d_{j_i} = (x_1, y_1, x_2, y_2, \ldots, x_{\mid\mid d_{j_i} \mid\mid -1}, y_{\mid\mid d_{j_i} \mid\mid -1}, s_i, t_i)$, for all $i \geq 1$. According to Lemma 6.3.39, $u_{j_i} = x_1 x_2 \cdots x_{\mid\mid d_{j_i} \mid\mid -1} s_i \in \mathrm{pref}(u_{j_{i+1}}) = \mathrm{pref}(x_1 x_2 \cdots x_{\mid\mid d_{j_{i+1}} \mid\mid -1} s_{i+1})$, for all $i \geq 1$, and thus $u = \lim_{n\to\infty} x_1 x_2 \cdots x_n$. Analogously we get $v = \lim_{n\to\infty} y_1 y_2 \cdots y_n$, and $w = \lim_{n\to\infty} x_1 y_1 x_2 y_2 \cdots x_n y_n$. Thus $w = x_1 y_1 x_2 y_2 \cdots$ with $x_1 \in \Delta^*$, $x_i \in \Delta^+$ for all $i \geq 2$, $y_i \in \Delta^+$ for all $i \geq 1$, $u = x_1 x_2 \cdots$, and $v = y_1 y_2 \cdots$. Hence $w \in u \mid\mid v$. □

The preceding two lemmata allow us to conclude that whenever the prefixes of an infinite word w are included in the shuffle of the prefixes of two words u and v that do not share a single letter, then w is a shuffle of u and v.

Lemma 6.3.41. *Let $u, v \in \Delta^\infty$ be such that alph$(u) \cap$ alph$(v) = \varnothing$ and let $w \in \Delta^\omega$. Then*

if pref$(w) \subseteq$ pref$(u) \mid\mid$ pref(v), then $w \in u \mid\mid v$.

Proof. Let $\mathrm{pref}(w) \subseteq \mathrm{pref}(u) \mid\mid \mathrm{pref}(v)$. Now consider two arbitrary consecutive prefixes of w. Thus for some $n \geq 0$ we have $w[n]$ and $w[n+1] = w[n]a$ such that $a \in \mathrm{alph}(u)$ or $a \in \mathrm{alph}(v)$. Since $\mathrm{pref}(w) \subseteq \mathrm{pref}(u) \mid\mid \mathrm{pref}(v)$, there are prefixes u_n and u_{n+1} of u, and prefixes v_n and v_{n+1} of v such that $w[n] \in u_n \mid\mid v_n$ and $w[n+1] \in u_{n+1} \mid\mid v_{n+1}$. Observe that $\#_a(w[n+1]) = \#_a(w[n]) + 1$. Moreover, for all $b \in \mathrm{alph}(u)$ and for all $c \in \mathrm{alph}(v)$ such that $b \neq a$ and $c \neq a$ we have $\#_b(w[n]) = \#_b(u_n) = \#_b(w[n+1]) = \#_b(u_{n+1})$ and

$\#_c(w[n]) = \#_c(v_n) = \#_c(w[n+1]) = \#_c(v_{n+1})$ because $w[n+1] = w[n]a$ and $\mathrm{alph}(u) \cap \mathrm{alph}(v) = \varnothing$.

Consequently, using the fact that u_{n+1} and u_n are both prefixes of u, and v_{n+1} and v_n are both prefixes of v we conclude that $u_{n+1} = u_n a$ and $v_{n+1} = v_n$ if $a \in \mathrm{alph}(u)$, and $v_{n+1} = v_n a$ and $u_{n+1} = u_n$ if $a \in \mathrm{alph}(v)$.

Now let d_n be a (u_n, v_n)-decomposition of $w[n]$. Then we have $d_n = (x_1, y_1, x_2, y_2, \ldots, x_k, y_k)$, with $k \geq 0$. We define a (u_{n+1}, v_{n+1})-decomposition of $w[n+1]$ as follows.

First let $a \in \mathrm{alph}(u)$. If $y_k = \lambda$, then $d_{n+1} = (x_1, y_1, x_2, y_2, \ldots, x_k a, y_k)$, whereas if $y_k \neq \lambda$, then we set $d_{n+1} = (x_1, y_1, x_2, y_2, \ldots, x_k, y_k, a, \lambda)$. In both cases we have $x_1 x_2 \cdots x_k a = u_n a = u_{n+1}$ and $y_1 y_2 \cdots y_k = v_n = v_{n+1}$. Moreover $x_1 y_1 x_2 y_2 \cdots x_k y_k a = w[n]a = w[n{+}1]$. Thus d_{n+1} is a (u_{n+1}, v_{n+1})-decomposition of $w[n+1]$ and d_n precedes d_{n+1}.

Next we let $a \in \mathrm{alph}(v)$. Now $d_{n+1} = (x_1, y_1, x_2, y_2, \ldots, x_k, y_k a)$. Since $x_1 x_2 \cdots x_k = u_n = u_{n+1}$ and $y_1 y_2 \cdots y_k a = v_n a = v_{n+1}$ are such that $x_1 y_1 x_2 y_2 \cdots x_k y_k a = w[n]a = w[n+1]$ we thus know that d_{n+1} is a (u_{n+1}, v_{n+1})-decomposition of $w[n+1]$, which is preceded by d_n.

Observe that the only decomposition of $w[0] = \lambda$ is $d_0 = (\lambda, \lambda)$. Hence we have defined an infinite (and unique) sequence of (u_i, v_i)-decompositions d_i of $w[i]$, $i \geq 0$, such that d_i precedes d_{i+1} for all $i \geq 0$. Hence from Lemmata 6.3.40 it follows that $w = \lim_{n\to\infty} w[n] \in \lim_{n\to\infty} u_n \mid\mid \lim_{n\to\infty} v_n = u \mid\mid v$. □

This result implies that in order to determine whether or not an infinite word is a shuffle of two (possibly infinite) words that do not share a single letter, it suffices to consider only the (finite!) prefixes of those words. Unfortunately, however, condition $\mathrm{alph}(u) \cap \mathrm{alph}(v) = \varnothing$ of Lemma 6.3.41 is necessary to prove that each prefix of w has a unique decomposition into prefixes of u and v. This is illustrated in the following example. We moreover show that there exist an infinite number of prefixes $w[n]$ with a decomposition that does not precede any decomposition of $w[n+1]$.

Example 6.3.42. Let $\Delta = \{a, b\}$. Let $u, v \in \Delta^\omega$ be such that $u = (a^3 b)^\omega$ and $v = b^\omega$. Clearly $\{a^3, a^3 b\} \subseteq \mathrm{pref}(u)$, $\{b^2, b^3\} \subseteq \mathrm{pref}(v)$, and $w = a^3 b^3 \in \mathrm{pref}(u) \mid\mid \mathrm{pref}(v)$. We thus note that $d_1 = (a^3, b^3)$ and $d_2 = (a^3 b, b^2)$ are decompositions of w.

Note that also $w' = wa = a^3 b^3 a \in \mathrm{pref}(u) \mid\mid \mathrm{pref}(v)$. The only decompositions of w' based on prefixes of u and v are $d' = (a^3 b, b^2, a, \lambda)$ and $d'' = (a^3, b^2, ba, \lambda)$. It is clear that d_1 does not precede d' nor does it precede d''. Hence w and $w' = wa$ are such that there exists a decomposition d_1 of w that does not precede any decomposition of w'. Note, however, that d' is preceded by d_2.

Let $j \geq 0$ and let $u_j = a^3(ba^3)^j \in \mathrm{pref}(u)$ and $v_j = b^3(b^3)^j \in \mathrm{pref}(v)$. Then clearly both $w_j = (a^3b^4)^j a^3b^3 \in \mathrm{pref}(u) \mid\mid \mathrm{pref}(v)$ and $w'_j = w_j a = (a^3b^4)^j a^3b^3a \in \mathrm{pref}(u) \mid\mid \mathrm{pref}(v)$. Now note that $d_j = (x_0, y_0, x_1, y_1, \ldots, x_j, y_j, a^3, b^3)$, where $x_i = a^3b$ and $y_i = b^3$ for all $0 \leq i \leq j$, is a (u_j, v_j)-decomposition of w_j. By the same reasoning as for the case $j = 0$ above it is however easy to see that there does not exist a decomposition of w'_j based on prefixes of u and v that is preceded by d_j. □

In order to generalize Lemma 6.3.41 by dropping the condition $\mathrm{alph}(u) \cap \mathrm{alph}(v) \neq \varnothing$ we need to be able to guarantee the following: if $u, v \in \Delta^\infty$, $w \in \Delta^\omega$, and $\mathrm{pref}(w) \subseteq \mathrm{pref}(u) \mid\mid \mathrm{pref}(v)$, then there exists an infinite sequence of (u_n, v_n)-decompositions of $w[n]$, with $n \geq 0$, preceding each other. With this in mind we now recall *König's Lemma*.

Lemma 6.3.43. *(König's Lemma) If G is an infinite finitely-branching rooted tree, then there exists an infinite path through G, starting in the root.* □

The subsequent definition of *limit-closed* languages allows us to first generalize Lemma 6.3.41 to languages and then to infer that the condition $\mathrm{alph}(u) \cap \mathrm{alph}(v) \neq \varnothing$ can — after all — indeed be dropped from Lemma 6.3.41.

Definition 6.3.44. *Let $K \subseteq \Delta^\infty$. Then*

> K *is* limit closed *if for all words $w_1 \leq w_2 \leq \cdots \in \mathit{pref}(K)$, $\lim_{n\to\infty} w_n \in K \cup \mathit{pref}(K)$.* □

Example 6.3.45. All singleton languages $\{u\}$ are limit closed. Also all finitary languages $L = \{\lambda, a, \ldots, a^n \mid n \geq 1\}$ over a unary alphabet are limit closed, whereas a^* is not limit closed due to the fact that $\lim_{n\to\infty} a^n = a^\omega \notin a^* \cup L$. However, $a^* \cup a^\omega$ and a^ω are limit closed. □

Lemma 6.3.46. *Let $K, L \subseteq \Delta^\infty$ be limit closed and let $w \in \Delta^\omega$. Then*

> *if $\mathit{pref}(w) \subseteq \mathit{pref}(K) \mid\mid \mathit{pref}(L)$, then $w \in K \mid\mid L$.*

Proof. Let $\mathrm{pref}(w) \subseteq \mathrm{pref}(K) \mid\mid \mathrm{pref}(L)$.

For $n \geq 0$, let $V_n = \{d \mid d \text{ is a } (u_n, v_n)\text{-decomposition of } w[n],\ u_n \in \mathrm{pref}(K), \text{ and } v_n \in \mathrm{pref}(L)\}$ be the set of all possible decompositions of the prefixes $w[n]$ of w. Note that $V_0 = \{(\lambda, \lambda)\}$ consists of the (λ, λ)-decomposition of $w[0] = \lambda$. Note furthermore that each V_n is finite, for $n \geq 0$, and that $V_n \cap V_{n'} = \varnothing$, for all $n > n' \geq 0$.

Consider the directly precedes relation $E = \{(d, d') \mid d \text{ directly precedes } d'\}$. Thus $E \subseteq \bigcup_{n\geq 1}(V_{n-1} \times V_n)$. Note that $G = (\bigcup_{n\geq 0} V_n, E)$ is a directed acyclic graph. It is sketched in Figure 6.7.

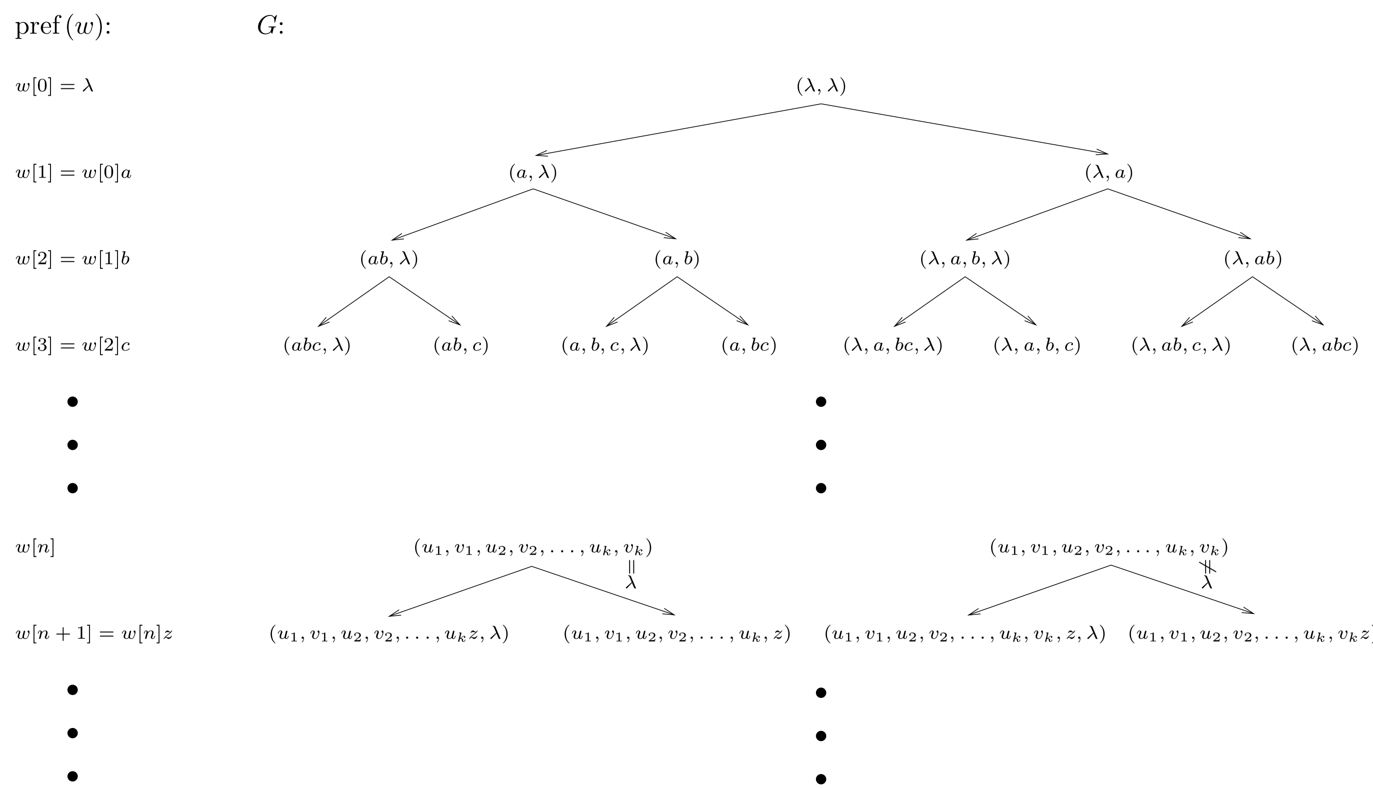


Fig. 6.7. Sketch of tree $G = (\bigcup_{n \geq 0} V_n, E)$.

Except for (λ, λ), every vertex of G has precisely one incoming edge. This can be seen as follows. The fact that $\mathrm{pref}(w) \subseteq \mathrm{pref}(K) \mid\mid \mathrm{pref}(L)$ implies that every vertex has at least one incoming edge, whereas the fact that for every decomposition of a prefix $w[n]$, $n \geq 1$, we can immediately distinguish the unique last symbol of $w[n]$, implies that every vertex has at most one incoming edge. Furthermore, from Definition 6.3.38 it follows that every vertex has at most two outgoing edges, depending on whether the symbol added to $w[n]$, $n \geq 0$, to obtain $w[n+1]$ "belongs" to a prefix from K or to a prefix from L. Hence G is an infinite finitely-branching rooted tree with root (λ, λ).

We can thus use König's Lemma to conclude that there exists an infinite path π through G, starting in the root (λ, λ). Let $\pi = (d_0, d_1, \dots)$. Then for all $n \geq 0$, d_n is a (u_n, v_n)-decomposition of $w[n]$ and $(d_n, d_{n+1}) \in E$. Hence from Lemma 6.3.40 it follows that $u = \lim\limits_{n\to\infty} u_n$, $v = \lim\limits_{n\to\infty} v_n$, and $w = \lim\limits_{n\to\infty} w_n$ exist, and $w \in u \mid\mid v$. Since K and L are limit closed this implies that $w \in K \mid\mid L$. $\square$

The statement of this lemma in general does not hold when K or L are not limit closed, as is shown next.

Example 6.3.47. Let $\Delta = \{a\}$ and let $w = a^\omega \in \Delta^\omega$. Let $K = a^* \subseteq \Delta^\infty$ and let $L = \{\lambda\} \subseteq \Delta^\infty$. Then clearly $\mathrm{pref}(w) = a^* = \mathrm{pref}(K) \mid\mid \mathrm{pref}(L)$, whereas $w = a^\omega \notin a^* = K \mid\mid L$. $\square$

Since all singleton languages are limit closed, we immediately obtain the following result.

Corollary 6.3.48. *Let $u, v \in \Delta^\infty$ and let $w \in \Delta^\omega$. Then*

if $\mathit{pref}(w) \subseteq \mathit{pref}(u) \mid\mid \mathit{pref}(v)$, then $w \in u \mid\mid v$. $\square$

Together with Theorem 6.3.21, this corollary and its preceding lemma imply the following result.

Theorem 6.3.49. *Let $u, v \in \Delta^\infty$, let $K, L \subseteq \Delta^\infty$ be limit closed, and let $w \in \Delta^\omega$. Then*

(1) $w \in u \mid\mid v$ if and only if $\mathit{pref}(w) \subseteq \mathit{pref}(u) \mid\mid \mathit{pref}(v)$, and

(2) $w \in K \mid\mid L$ if and only if $\mathit{pref}(w) \subseteq \mathit{pref}(K) \mid\mid \mathit{pref}(L)$. $\square$

We have thus been able to express the shuffles of possibly infinite words in terms of the shuffles of finite prefixes of those possibly infinite words. One more result now suffices to prove the associativity of shuffling.

Corollary 6.3.50. *Let $v, w \in \Delta^\infty$. Then*

$v \mid\mid w$ is limit closed.

Proof. Let $y_1 \leq y_2 \leq \cdots \in \operatorname{pref}(v \mid\mid w)$ and let $y = \lim_{n\to\infty} y_n$. Since for all $x \in \operatorname{pref}(y)$, there exists an $i \geq 0$ such that $x \in \operatorname{pref}(y_i) \in \operatorname{pref}(\operatorname{pref}(v \mid\mid w)) = \operatorname{pref}(v \mid\mid w)$, it follows that $\operatorname{pref}(y) \subseteq \operatorname{pref}(v \mid\mid w)$. Consequently, we distinguish two cases.

If $y \in \Delta^*$, then $y \in \operatorname{pref}(v \mid\mid w)$.

If $y \in \Delta^\omega$, then by Theorem 6.3.49(1), $y \in v \mid\mid w$.

Hence $y \in v \mid\mid w \cup \operatorname{pref}(v \mid\mid w)$ and $v \mid\mid w$ is thus limit closed. □

Theorem 6.3.51. *Let $u, v, w \in \Delta^\infty$ and let $L_1, L_2, L_3 \subseteq \Delta^\infty$. Then*

(1) $\{u\} \mid\mid (v \mid\mid w) = (u \mid\mid v) \mid\mid \{w\}$ and

(2) $L_1 \mid\mid (L_2 \mid\mid L_3) = (L_1 \mid\mid L_2) \mid\mid L_3$.

Proof. (1) Let $x \in \{u\} \mid\mid (v \mid\mid w)$.

If $x \in \Delta^*$, then Definition 6.3.1 implies that $u, v, w \in \Delta^*$. Consequently, by Corollary 6.3.33(1), $x \in (u \mid\mid v) \mid\mid \{w\}$.

If $x \in \Delta^\omega$, then since we know that $\{u\}$ and $v \mid\mid w$ are limit closed, Theorem 6.3.49(2) implies that $\operatorname{pref}(x) \subseteq \operatorname{pref}(\{u\}) \mid\mid \operatorname{pref}(v \mid\mid w)$. Hence, by Theorem 6.3.21(1), $\operatorname{pref}(x) \subseteq \operatorname{pref}(\{u\}) \mid\mid (\operatorname{pref}(v) \mid\mid \operatorname{pref}(w))$. Then Corollary 6.3.33(2) implies that $\operatorname{pref}(x) \subseteq (\operatorname{pref}(u) \mid\mid \operatorname{pref}(v)) \mid\mid \operatorname{pref}(\{w\})$ and from Theorem 6.3.21(1) we obtain $\operatorname{pref}(x) \subseteq \operatorname{pref}(u \mid\mid v) \mid\mid \operatorname{pref}(\{w\})$. Finally, using the fact that $u \mid\mid v$ and $\{w\}$ are limit closed, Theorem 6.3.49(2) implies that $x \in (u \mid\mid v) \mid\mid \{w\}$.

(2) Analogous to the proof of Theorem 6.3.32(2). □

6.3.4 Conclusion

The associativity of (fairly) shuffling (cf. Theorems 6.3.32 and 6.3.51) directly implies that the order in which we (fairly) shuffle a number of languages is irrelevant, i.e. $L_1 \mid\mid\mid L_2 \mid\mid\mid \cdots \mid\mid\mid L_n$ and $L_1 \mid\mid L_2 \mid\mid \cdots \mid\mid L_n$ unambiguously define the fair shuffle and shuffle, respectively, of the languages $L_1, L_2, \ldots, L_n$, for an $n \geq 1$. It is thus not necessary to put any brackets in these expressions and we will henceforth refrain from doing so. Using also the commutativity of (fairly) shuffling, we may introduce the following shorthand notations for such *n-ary (fair) shuffles*.

Notation 12. *We denote the fair shuffle $L_1 \mid\mid\mid L_2 \mid\mid\mid \cdots \mid\mid\mid L_n$ and the shuffle $L_1 \mid\mid L_2 \mid\mid \cdots \mid\mid L_n$ of the languages $L_1, L_2, \ldots, L_n$, for an $n \geq 1$, by $\mid\mid\mid_{i\in[n]} L_i$ and $\mid\mid_{i\in[n]} L_i$, respectively.* □

6.4 Synchronized Shuffles

In this section we generalize the basic shuffle by defining *synchronized shuffles*. Rather than freely interleaving the occurrences of the letters in the words being shuffled, some letters may now be subject to "synchronization". This means that occurrences of those letters in different words are now combined into one occurrence. The resulting word thus has a "backbone" consisting of occurrences of synchronized letters. As a preliminary example, consider the words *wev* and *ave*. If we assume that the letter v needs to be synchronized, then *weave* is a *synchronized shuffle on* v *of* *wev* and *ave*. Its backbone consists of only one element, viz. v. We see that those letters occurring on the left (right) side of v in the original words occur on the left (right) side of v in *weave* as well. Note that *weave* is not an ordinary shuffle of *wev* and *ave*.

As was the case for shuffles, also the idea underlying synchronized shuffles is not new. Instead, it appears in numerous disguises throughout the computer science literature. The oldest reference — once again to the best of our knowledge — to this idea is the *concurrent composition* $P \oplus Q$ *of synchronizing processes* P and Q defined in [Kim76]. Within formal language theory, a slightly adapted version of the idea was introduced in [DeS84] as the '*produit de mixage*' $K \sqcap\!\sqcap L$ of two languages K and L. This operation was renamed *synchronized shuffle* in [LR99]. In the context of process algebra, finally, two further slightly adapted versions of the idea were introduced in [vdS85] as the *weave* $T \, \underline{w} \, U$ of two words T and U, and in [Ros97] as the *alphabetized parallel composition* $P \;{}_X\underline{\|}_Y\; Q$ of processes P and Q given alphabets X and Y. We will soon see, however, that the synchronized shuffles we define here are more general than any of these operations from the literature. In particular, we define two variants of synchronized shuffles: the *fully synchronized shuffle* and the *relaxed synchronized shuffle*, both obtained by varying the alphabet of letters to be synchronized.

Given two words over two given (possibly different) alphabets, a fully synchronized shuffle requires all letters in the intersection of these two alphabets to be synchronized, while a relaxed synchronized shuffle requires only a specified subset of the letters in this intersection to be synchronized. Both synchronized shuffles are thus defined with respect to two alphabets. We continue our example by again considering the words *wev* and *ave*. Assume that *wev* is a word over the alphabet $\{w, e, v\}$ and that *ave* is a word over the alphabet $\{a, v, e\}$. Then a *fully synchronized shuffle of* *wev* and *ave* *w.r.t.* $\{w, e, v\}$ and $\{a, v, e\}$ does not exist due to the fact that e and v cannot form one backbone respecting both the order *ev* from *wev* and the order *ve* from *ave*. However, a *relaxed synchronized shuffle on* $\{e\}$ *of* *wev* and *ave* *w.r.t.* $\{w, e, v\}$ and $\{a, v, e\}$ does exist and contains, e.g., *wavev*.

We begin by formally defining the most general synchronized shuffle, in terms of which we consequently define the two variants just discussed — complete with more elaborate examples. Along the way we will compare our synchronized shuffles to the ones from the literature. Subsequently we present a few of their basic properties. Since synchronized shuffles are defined on the basis of the ordinary shuffle, many observations from the previous section continue to hold (with trivial adaptions). We will not draw all such implications, but rather provide a series of connections between the various types of (synchronized) shuffles. Finally, we prove that all three types of synchronized shuffles satisfy notions of commutativity and associativity.

6.4.1 Definitions

We start by defining *synchronized shuffles* as a generalization of the shuffles of the previous section. Given an alphabet Γ and two words u and v, in a synchronized shuffle u and v synchronize on letters from Γ, while all occurrences of other letters are shuffled.

Definition 6.4.1. *Let $u, v \in \Delta^\infty$ and let Γ be an alphabet. Then*

a word $w \in \Delta^\infty$ is a synchronized shuffle *(*S-shuffle *for short)* on Γ *of u and v if one of the following two cases holds. Either*

(1) $w \in (u_1 \mathbin{||} v_1)x_1(u_2 \mathbin{||} v_2)x_2 \cdots x_{n-1}(u_n \mathbin{||} v_n)$, where for some $n \geq 1$, $u_1, u_2, \ldots, u_{n-1}, v_1, v_2, \ldots, v_{n-1} \in (\Delta \setminus \Gamma)^*$, $u_n, v_n \in (\Delta \setminus \Gamma)^\infty$, and $x_1, x_2, \ldots, x_{n-1} \in \Gamma$ are such that $u = u_1x_1u_2x_2 \cdots x_{n-1}u_n$ and $v = v_1x_1v_2x_2 \cdots x_{n-1}v_n$, or

(2) $w \in (u_1 \mathbin{||} v_1)x_1(u_2 \mathbin{||} v_2)x_2 \cdots$, where $u_1, u_2, \ldots, v_1, v_2, \cdots \in (\Delta \setminus \Gamma)^*$, and $x_1, x_2, \cdots \in \Gamma$ are such that $u = u_1x_1u_2x_2 \cdots$ and $v = v_1x_1v_2x_2 \cdots$.

This S-shuffle w on Γ is called fair *if in case (1) $(u_n \mathbin{||} v_n)$ is fair or if case (2) holds.* □

The sequence $\mathrm{pres}_\Gamma(w)$ is called the *backbone* of w. Note that in case (1) the S-shuffle w has a finite backbone $x_1x_2 \cdots x_{n-1}$, while in case (2) it has an infinite backbone $x_1x_2 \cdots$.

For $u, v \in \Delta^\infty$ the language consisting of all *(fair) S-shuffles* on Γ of u and v is denoted by $u \mathbin{||}^\Gamma v$ ($u \mathbin{|||}^\Gamma v$) and is defined as $u \mathbin{||}^\Gamma v = \{w \in \Delta^\infty \mid w \text{ is an S-shuffle on } \Gamma \text{ of } u \text{ and } v\}$ and $u \mathbin{|||}^\Gamma v = \{w \in \Delta^\infty \mid w \text{ is a fair S-shuffle on } \Gamma \text{ of } u \text{ and } v\}$, respectively.

For $L_1, L_2 \subseteq \Delta^\infty$ the *(fair) S-shuffle* on Γ of L_1 and L_2 is denoted by $L_1 \mathbin{||}^\Gamma L_2$ ($L_1 \mathbin{|||}^\Gamma L_2$) and is defined as the language consisting of all (fair)

S-shuffles on Γ of a word from L_1 and a word from L_2. Thus $L_1 \mid\mid^{\Gamma} L_2 = \{w \in u \mid\mid^{\Gamma} v \mid u \in L_1,\ v \in L_2\} = \bigcup_{u \in L_1,\, v \in L_2}(u \mid\mid^{\Gamma} v)$ and $L_1 \mid\mid\mid^{\Gamma} L_2 = \bigcup_{u \in L_1,\, v \in L_2}(u \mid\mid\mid^{\Gamma} v)$, respectively.

Example 6.4.2. (Example 6.3.2 continued) Recall that $u = abc$ and $v = cd$. Now $u \mid\mid^{\{c\}} v = u \mid\mid\mid^{\{c\}} v = \{abcd\}$, whereas $u \mid\mid^{\{b,c\}} v = u \mid\mid\mid^{\{b,c\}} v = \varnothing$.

Recall that $w_1 = a^{\omega}$. Now $w_1 \mid\mid^{\{a\}} a = w_1 \mid\mid\mid^{\{a\}} a = \varnothing$ and $w_1 \mid\mid^{\{a\}} w_1 = w_1 \mid\mid\mid^{\{a\}} w_1 = \{a^{\omega}\}$.

Finally, recall that $\Delta = \{a, b, c, d\}$. Let $w_{12} = (ab)^{\omega} \in \Delta^{\omega}$ and let $w_{21} = (ba)^{\omega} \in \Delta^{\omega}$. Then we have $w_{12} \mid\mid^{\{a\}} w_{21} = w_{12} \mid\mid\mid^{\{a\}} w_{21} = \{(bab)^{\omega}\}$, whereas $w_{12} \mid\mid^{\{a,b\}} w_{21} = w_{12} \mid\mid\mid^{\{a,b\}} w_{21} = \varnothing$. □

From Definition 6.4.1 we furthermore obtain that the fair S-shuffle on an alphabet Γ of languages is included in the S-shuffle on Γ of these languages.

We now show that S-shuffles are indeed a generalization of both the concurrent composition as defined in [Kim76] and the 'produit de mixage' as defined in [DeS84] (and later renamed synchronized shuffle in [LR99]). If we syntactically restrict an S-shuffle on an alphabet Γ of languages $L_1, L_2 \subseteq \Delta^*$ to the case that $\Gamma \subseteq \Delta$, then we obtain exactly the concurrent composition operation defined in [Kim76]. If, on the other hand, we define the *alphabet* $\mathrm{alph}(L)$ of a language L as $\mathrm{alph}(L) = \bigcup_{w \in L} \mathrm{alph}(w)$ and allow infinite words in L_1 and L_2, then $L_1 \mid\mid\mid^{\mathrm{alph}(L_1) \cap \mathrm{alph}(L_2)} L_2$ is exactly the 'produit de mixage' of L_1 and L_2 as defined in [DeS84] (which in [LR99] is restricted to finitary languages and renamed synchonized shuffle).

We proceed by defining the *fully synchronized shuffle* as a special case of the synchronized shuffle. Given a word u over Δ_1 and a word v over Δ_2, in a fully synchronized shuffle u and v synchronize on letters from $\Delta_1 \cap \Delta_2$, while all occurrences of other letters are again shuffled. Limited to finite words, the fully synchronized shuffle is exactly the weave operation defined in [vdS85] in the context of process algebra. By allowing infinite words, the fully synchronized shuffle is thus more general than the weave operation.

Definition 6.4.3. *Let $u \in \Delta_1^{\infty}$ and let $v \in \Delta_2^{\infty}$. Then*

> *a word $w \in (\Delta_1 \cup \Delta_2)^{\infty}$ is a* fully synchronized shuffle *(*fS-shuffle *for short) of u and v w.r.t. Δ_1 and Δ_2 if w is an S-shuffle on $\Delta_1 \cap \Delta_2$ of u and v.*

This fS-shuffle of u and v w.r.t. Δ_1 and Δ_2 is called fair *if w is a fair S-shuffle on $\Delta_1 \cap \Delta_2$ of u and v.* □

For $u \in \Delta_1^{\infty}$ and $v \in \Delta_2^{\infty}$ the language consisting of all *(fair) fS-shuffles* of u and v w.r.t. Δ_1 and Δ_2 is denoted by $u \ {}_{\Delta_1}\!\underline{\mid\mid}_{\Delta_2}\ v$ ($u \ {}_{\Delta_1}\!\underline{\mid\mid\mid}_{\Delta_2}\ v$) and is defined as $u \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid w$ is an fS-shuffle of u and v w.r.t. Δ_1 and $\Delta_2\}$ and $u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid w$ is a fair fS-shuffle of u and v w.r.t. Δ_1 and $\Delta_2\}$, respectively.

For $L_1 \subseteq \Delta_1^\infty$ and $L_2 \subseteq \Delta^\infty$ the *(fair) fS-shuffle* of L_1 and L_2 w.r.t. Δ_1 and Δ_2 is denoted by $L_1 \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; L_2$ ($L_1 \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; L_2$) and is defined as the language consisting of all (fair) fS-shuffles of a word from L_1 and a word from L_2 w.r.t. Δ_1 and Δ_2. Thus $L_1 \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; L_2 = \{w \in u \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; v \mid u \in L_1,\, v \in L_2\} = \bigcup_{u \in L_1,\, v \in L_2}(u \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; v)$ and $L_1 \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; L_2 = \bigcup_{u \in L_1,\, v \in L_2}(u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v)$, respectively.

Example 6.4.4. (Example 6.4.2 continued) Now $u \;{}_{\Delta}\underline{||}_{\Delta}\; v = u \;{}_{\Delta}\underline{|||}_{\Delta}\; v = \varnothing$. Next let $\Delta_1 = \{a, b, c\}$ and let $\Delta_2 = \{c, d\}$. Consequently, let $u = abc \in \Delta_1^*$ and let $v = cd \in \Delta_2^*$. Then $u \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; v = u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v = \{abcd\} = u \;|||^{\{c\}}\; v = u \;||^{\{c\}}\; v$.

We moreover have $w_1 \;{}_{\Delta}\underline{||}_{\Delta}\; a = w_1 \;{}_{\Delta}\underline{|||}_{\Delta}\; a = \varnothing$, with $a \in \Delta^*$, and $w_1 \;{}_{\Delta}\underline{||}_{\Delta}\; w_1 = w_1 \;{}_{\Delta}\underline{|||}_{\Delta}\; w_1 = \{a^\omega\} = w_1 \;|||^{\{a\}}\; w_1 = w_1 \;||^{\{a\}}\; w_1$. Recall that $w_2 = b^\omega \in \Delta^\infty$ and hence $w_1 \;{}_{\Delta}\underline{||}_{\Delta}\; w_2 = w_1 \;{}_{\Delta}\underline{|||}_{\Delta}\; w_2 = \varnothing$. Next let $\Delta_a = \{a\}$ and let $\Delta_b = \{b\}$. Consequently, let $w_1 = a^\omega \in \Delta_a^\infty$ and let $w_2 = b^\omega \in \Delta_b^\infty$. Then $w_1 \;{}_{\Delta_a}\underline{||}_{\Delta_b}\; w_2 = w_1 \;||\; w_2$ and $w_1 \;{}_{\Delta_a}\underline{|||}_{\Delta_b}\; w_2 = w_1 \;|||\; w_2$.

Finally, $w_{12} \;{}_{\Delta}\underline{||}_{\Delta}\; w_{21} = w_{12} \;{}_{\Delta}\underline{|||}_{\Delta}\; w_{21} = \varnothing$. □

Finally we define also the *relaxed synchronized shuffle* as a special case of the synchronized shuffle. Given an alphabet Γ, a word u over Δ_1, and a word v over Δ_2, in a relaxed synchronized shuffle u and v synchronize on letters from $\Gamma \cap \Delta_1 \cap \Delta_2$, while all occurrences of other letters are once again shuffled.

Definition 6.4.5. *Let $u \in \Delta_1^\infty$, let $v \in \Delta_2^\infty$, and let Γ be an alphabet. Then*

> *a word $w \in (\Delta_1 \cup \Delta_2)^\infty$ is a* relaxed synchronized shuffle *(*rS-shuffle *for short)* on *Γ of u and v w.r.t. Δ_1 and Δ_2 if w is an S-shuffle on $\Gamma \cap \Delta_1 \cap \Delta_2$ of u and v.*

This rS-shuffle on Γ of u and v w.r.t. Δ_1 and Δ_2 is called fair *if w is a fair S-shuffle on $\Gamma \cap \Delta_1 \cap \Delta_2$ of u and v.* □

For $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$ the language consisting of all *(fair) rS-shuffles* on Γ of u and v w.r.t. Δ_1 and Δ_2 is denoted by $u \;{}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\; v$ ($u \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; v$) and is defined as $u \;{}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\; v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid w$ is an rS-shuffle on Γ of u and v w.r.t. Δ_1 and $\Delta_2\}$ and $u \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid w$ is a fair rS-shuffle on Γ of u and v w.r.t. Δ_1 and $\Delta_2\}$, respectively.

For $L_1 \subseteq \Delta_1^\infty$ and $L_2 \subseteq \Delta^\infty$ the *(fair) rS-shuffle* on Γ of L_1 and L_2 w.r.t. Δ_1 and Δ_2 is denoted by $L_1 \;{}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\; L_2$ ($L_1 \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; L_2$) and is defined as the language consisting of all (fair) rS-shuffles on Γ of a word from L_1 and a word from L_2 w.r.t. Δ_1 and Δ_2. Thus $L_1 \ {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\ L_2 = \{w \in u \ {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\ v \mid u \in L_1,\ v \in L_2\} = \bigcup_{u \in L_1,\, v \in L_2}(u \ {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\ v)$ and $L_1 \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\ L_2 = \bigcup_{u \in L_1,\, v \in L_2}(u \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\ v)$, respectively.

Example 6.4.6. (Example 6.4.4 continued) Now $u \ {}_{\Delta}\underline{||}^{\{c\}}_{\Delta}\ v = u \ {}_{\Delta}\underline{|||}^{\{c\}}_{\Delta}\ v = \{abcd\}$, whereas $u \ {}_{\Delta}\underline{||}^{\{b,c\}}_{\Delta}\ v = u \ {}_{\Delta}\underline{|||}^{\{b,c\}}_{\Delta}\ v = \varnothing$. Furthermore, $u \ {}_{\Delta_1}\underline{||}^{\{c\}}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{|||}^{\{c\}}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{|||}^{\{b,c\}}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{||}^{\{b,c\}}_{\Delta_2}\ v = \{abcd\} = u \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{|||}_{\Delta_2}\ v = u \ |||^{\{c\}}\ v = u \ ||^{\{c\}}\ v$.

We moreover have $w_1 \ {}_{\Delta}\underline{||}^{\{a\}}_{\Delta}\ a = w_1 \ {}_{\Delta}\underline{|||}^{\{a\}}_{\Delta}\ a = \varnothing$, with $a \in \Delta^*$, and $w_1 \ {}_{\Delta}\underline{||}^{\{a\}}_{\Delta}\ w_1 = w_1 \ {}_{\Delta}\underline{|||}^{\{a\}}_{\Delta}\ w_1 = \{a^\omega\} = w_1 \ {}_{\Delta}\underline{||}_{\Delta}\ w_1 = w_1 \ {}_{\Delta}\underline{|||}_{\Delta}\ w_1 = w_1 \ |||^{\{a\}}\ w_1 = w_1 \ ||^{\{a\}}\ w_1$. We also have $w_1 \ {}_{\Delta}\underline{||}^{\{a\}}_{\Delta}\ w_2 = w_1 \ {}_{\Delta}\underline{|||}^{\{a\}}_{\Delta}\ w_2 = \varnothing$, $w_1 \ {}_{\Delta_a}\underline{||}^{\{a\}}_{\Delta_b}\ w_2 = w_1 \ ||\ w_2$, and $w_1 \ {}_{\Delta_a}\underline{|||}^{\{a\}}_{\Delta_b}\ w_2 = w_1 \ |||\ w_2$.

Finally, here $w_{12} \ {}_{\Delta}\underline{||}^{\{a\}}_{\Delta}\ w_{21} = w_{12} \ {}_{\Delta}\underline{|||}^{\{a\}}_{\Delta}\ w_{21} = \{(bab)^\omega\}$, whereas $w_{12} \ {}_{\Delta}\underline{||}^{\{a,b\}}_{\Delta} w_{21} = w_{12} \ {}_{\Delta}\underline{|||}^{\{a,b\}}_{\Delta} w_{21} = \varnothing$. □

We now take a closer look at the three synchronized shuffles just introduced. We immediately note that the rS-shuffle can be considered to lie inbetween the S-shuffle and the fS-shuffle. In fact, the following results follow directly from Definitions 6.4.1, 6.4.3, and 6.4.5.

Lemma 6.4.7. *Let $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$. Let $K \subseteq \Delta_1^\infty$ and $L \subseteq \Delta_2^\infty$. Let Γ be an alphabet. Then*

(1) if $\Gamma \subseteq \Delta_1 \cap \Delta_2$, then $u \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\ v = u \ |||^{\Gamma}\ v$, $u \ {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\ v = u \ ||^{\Gamma}\ v$, $K \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\ L = K \ |||^{\Gamma}\ L$, and $K \ {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\ L = K \ ||^{\Gamma}\ L$, and

(2) if $\Gamma \supseteq \Delta_1 \cap \Delta_2$, then $u \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{|||}_{\Delta_2}\ v$, $u \ {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ v$, $K \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\ L = K \ {}_{\Delta_1}\underline{|||}_{\Delta_2}\ L$, and $K \ {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\ L = K \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ L$. □

We continue by pointing out that for arbitrary alphabets Δ_1, Δ_2, and Γ, both $u \ {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\ v$ and $u \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ v$ are undefined if either $u \notin \Delta_1^\infty$ or $v \notin \Delta_2^\infty$.

Finally, we show how this section's synchronized shuffles are related to the shuffle of the previous section. From Definition 6.4.1 we immediately obtain that the S-shuffle is indeed a generalization of the shuffle.

Lemma 6.4.8. *Let $u, v \in \Delta^\infty$ and let $K, L \subseteq \Delta^\infty$. Then*

(1) $u \ |||^{\varnothing}\ v = u \ |||\ v$ and $u \ ||^{\varnothing}\ v = u \ ||\ v$, and

(2) $K \ |||^{\varnothing}\ L = K \ |||\ L$ and $K \ ||^{\varnothing}\ L = K \ ||\ L$. □

Together with Example 6.3.2, this lemma implies that the inclusions of the fair S-shuffle on an alphabet Γ of languages in the S-shuffle on Γ of these languages may be proper. Furthermore, an S-shuffle on an alphabet Γ of languages is always fair in case both languages are finitary.

Moreover, the rS-shuffle degenerates to the shuffle if there are no letters to synchronize on.

Lemma 6.4.9. *Let $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$. Let $K \subseteq \Delta_1^\infty$ and $L \subseteq \Delta_2^\infty$. Then*

(1) $u \;{}_{\Delta_1}\underline{|||}^{\varnothing}_{\Delta_2}\; v = u \;|||^{\varnothing}\; v = u \;|||\; v$ and $u \;{}_{\Delta_1}\underline{||}^{\varnothing}_{\Delta_2}\; v = u \;||^{\varnothing}\; v = u \;||\; v$, and

(2) $K \;{}_{\Delta_1}\underline{|||}^{\varnothing}_{\Delta_2}\; L = K \;|||^{\varnothing}\; L = K \;|||\; L$ and $K \;{}_{\Delta_1}\underline{||}^{\varnothing}_{\Delta_2}\; L = K \;|||^{\varnothing}\; L = K \;||\; L$. □

Similarly, the fS-shuffle is a generalization of the shuffle in case of disjoint alphabets.

Lemma 6.4.10. *Let $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$. Let $K \subseteq \Delta_1^\infty$ and $L \subseteq \Delta_2^\infty$. Let $\Delta_1 \cap \Delta_2 = \varnothing$. Then*

(1) $u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v = u \;|||^{\varnothing}\; v = u \;|||\; v$ and $u \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; v = u \;||^{\varnothing}\; v = u \;||\; v$, and

(2) $K \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; L = K \;|||^{\varnothing}\; L = K \;|||\; L$ and $K \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; L = K \;||^{\varnothing}\; L = K \;||\; L$. □

6.4.2 Basic Observations

We have seen that a (fair) shuffle of two words always exists. From Example 6.4.2 we however conclude that a (fair) S-shuffle of two nonempty words need not exist. In fact, we have the following result.

Lemma 6.4.11. *Let $u, v \in \Delta^\infty$ and let Γ be an alphabet. Then*

(1) for all $w \in u \;||^{\Gamma}\; v$, $pres_\Gamma(w) = pres_\Gamma(u) = pres_\Gamma(v)$, and

(2) $u \;||^{\Gamma}\; v = \varnothing$ if and only if $pres_\Gamma(u) \neq pres_\Gamma(v)$.

Proof. (1) This follows immediately from Definition 6.4.1.

(2) (If) Let $u \;||^{\Gamma}\; v \neq \varnothing$. Then (1) implies that $\mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v)$.

(Only if) Let $\mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v) = w$. According to Definition 6.4.1 we thus need to distinguish two cases.

If there exists an $n \geq 0$ such that $w = x_1 x_2 \cdots x_n$, with $x_i \in \Gamma$ for all $i \in [n]$, then it must be the case that $u = u_1 x_1 u_2 x_2 \cdots x_n u_{n+1}$ and $v = v_1 x_1 v_2 x_2 \cdots x_n v_{n+1}$, with $u_i, v_i \in (\Delta \backslash \Gamma)^*$ for all $i \in [n]$ and $u_{n+1}, v_{n+1} \in$

$(\Delta \setminus \Gamma)^\infty$. Hence $u \mathbin{||}^\Gamma v = (u_1 \mathbin{||} v_1)x_1(u_2 \mathbin{||} v_2)x_2 \cdots x_n(u_{n+1} \mathbin{||} v_{n+1}) \neq \varnothing$ because for all $i \in [n+1]$, $u_i \mathbin{||} v_i \neq \varnothing$.

If $w = x_1x_2\cdots$, with $x_i \in \Gamma$ for all $i \geq 1$, then it must be the case that $u = u_1x_1u_2x_2\cdots$ and $v = v_1x_1v_2x_2\cdots$, with $u_i, v_i \in (\Delta \setminus \Gamma)^*$ for all $i \geq 1$. Hence $u \mathbin{||}^\Gamma v = (u_1 \mathbin{||} v_1)x_1(u_2 \mathbin{||} v_2)x_2 \cdots \neq \varnothing$ because for all $i \geq 1$, $u_i \mathbin{||} v_i \neq \varnothing$. □

We have also seen that the only (fair) shuffle of an arbitrary word and the empty word is the given word itself. Due to the requirement of a matching backbone, we immediately conclude that this in general does not hold when any of the (fair) synchronized shuffles is considered.

In Lemma 6.3.10, finally, we have seen that the length of every word in the shuffle of two finite words equals the sum of the lengths of those two words. Any synchronized shuffle of two finite words, however, may be a word of length less than the sum of the lengths of those two words. This is due to the fact that each letter from the synchronization alphabet must occur in both words being shuffled, while it occurs only once in the backbone of each synchronized shuffle of those words.

In the remainder of this subsection we seek to express the S-shuffles of possibly infinite words in terms of the S-shuffles of their finite prefixes. We begin by considering the case in which two words that are S-shuffled share a finite backbone (cf. Definition 6.4.1(1)). In such words u and v we can thus distinguish initial prefixes u_1 and v_1 ending with the last letter of the finite backbone, and suffixes u_2 and v_2 containing no more letters from the alphabet of the backbone. It is clear that elements of the S-shuffle of u and v then consist of a prefix that is part of the S-shuffle of u_1 and v_1 and a suffix that is part of the shuffle of u_2 and v_2. This leads to the following result.

Lemma 6.4.12. *Let Γ be an alphabet, let $u_1, v_1 \in ((\Delta \setminus \Gamma)^*\Gamma)^*$, and let $u_2, v_2 \in (\Delta \setminus \Gamma)^\infty$. Then*

(1) $(u_1 \mathbin{|||}^\Gamma v_1)(u_2 \mathbin{|||} v_2) = u_1u_2 \mathbin{|||}^\Gamma v_1v_2$ *and*

(2) $(u_1 \mathbin{||}^\Gamma v_1)(u_2 \mathbin{||} v_2) = u_1u_2 \mathbin{||}^\Gamma v_1v_2$. □

Note that this lemma resembles Lemma 6.3.14. The main difference between the two lemmata is the fact that the statements of Lemma 6.4.12 consist of equalities rather than inclusions from left to right only. The reason lies in the fact that the application of Lemma 6.4.12 is limited to prefixes which end at a predetermined position, viz. at the end of the backbone (which thus dictates the structure of all S-shuffles).

Lemma 6.4.12 consequently allows us to conclude that whenever the prefixes of an infinite word w are included in the S-shuffle of the prefixes of two words u and v sharing a finite backbone, then w is an element of the S-shuffle of u and v. In fact we prove a more general statement, immediately for prefixes of limited-closed languages (cf. Corollary 6.3.48 and Theorem 6.3.49).

Lemma 6.4.13. *Let $K, L \subseteq \Delta^\infty$ be limit closed, let Γ be an alphabet, and let $w = w_1 w_2$ be such that $w_1 \in ((\Delta \setminus \Gamma)^* \Gamma)^*$ and $w_2 \in (\Delta \setminus \Gamma)^\omega$. Then*

if $\mathit{pref}(w) \subseteq \mathit{pref}(K) \mathbin{||^\Gamma} \mathit{pref}(L)$, then $w \in K \mathbin{||^\Gamma} L$.

Proof. Let $\mathrm{pref}(w) \subseteq \mathrm{pref}(K) \mathbin{||^\Gamma} \mathrm{pref}(L)$. Then there exist an $n \geq 1$, $u_i \in \mathrm{pref}(K)$ and $v_i \in \mathrm{pref}(L)$ such that $w_1 \in u_i \mathbin{||^\Gamma} v_i$, for all $i \in [n]$. Note that according to Definition 6.4.1, all $u_i, v_i \in ((\Delta \setminus \Gamma)^* \Gamma)^*$. For all $i \in [n]$, let $K_{u_i} = \{u \in (\Delta \setminus \Gamma)^* \mid u_i u \in K\}$ and let $L_{v_i} = \{v \in (\Delta \setminus \Gamma)^* \mid v_i v \in L\}$.

Let $z \in \mathrm{pref}(w_2)$ and consider the word $w_1 z$. Thus $w_1 z \in \mathrm{pref}(K) \mathbin{||^\Gamma} \mathrm{pref}(L)$ because $w_1 z \in \mathrm{pref}(w)$. Hence there exist $u \in \mathrm{pref}(K)$ and $v \in \mathrm{pref}(L)$ such that $w_1 z \in u \mathbin{||^\Gamma} v$. Again by Definition 6.4.1 we know that $u = u'u''$ and $v = v'v''$, with $u', v' \in ((\Delta \setminus \Gamma)^* \Gamma)^*$ and $u'', v'' \in (\Delta \setminus \Gamma)^*$, and $w_1 \in u' \mathbin{||^\Gamma} v'$. Hence there exists an $i \in [n]$ such that $u' = u_i$ and $v' = v_i$. This implies that $w_1 z \in u_i \mathrm{pref}(K_{u_i}) \mathbin{||^\Gamma} v_i \mathrm{pref}(K_{v_i})$. Consequently, by Lemma 6.4.12(2), $\mathrm{pref}(w_2) \subseteq \bigcup_{i \in [n]} (\mathrm{pref}(K_{u_i}) \mathbin{||^\Gamma} \mathrm{pref}(L_{v_i})) = \bigcup_{i \in [n]} (\mathrm{pref}(K_{u_i}) \mathbin{||} \mathrm{pref}(L_{v_i}))$ (the equality follows because $\mathrm{pref}(K_{u_i})$ and $\mathrm{pref}(L_{v_i})$, with $i \in [n]$, do not contain letters from Γ).

Since the number of pairs u_i and v_i, with $i \in [n]$, is finite, it must be the case that there exists a $j \in [n]$ such that for each $z \in \mathrm{pref}(w_2)$ there exists a prefix z' of w_2 such that $z < z'$ and for which $z' \in \mathrm{pref}(K_{u_j}) \mathbin{||} \mathrm{pref}(L_{v_j})$ and thus $z \in \mathrm{pref}(K_{u_j}) \mathbin{||} \mathrm{pref}(L_{v_j})$. Hence $\mathrm{pref}(w_2) \subseteq \mathrm{pref}(K_{u_j}) \mathbin{||} \mathrm{pref}(L_{v_j})$. Since K and L are limit closed, so are K_{u_j} and L_{v_j}. Lemma 6.3.46 then implies that $w_2 \in K_{u_j} \mathbin{||} L_{v_j}$. Hence with Lemma 6.4.12(2) we obtain $w = w_1 w_2 \in (u_j \mathbin{||^\Gamma} v_j)(K_{u_j} \mathbin{||} L_{v_j}) = u_j K_{u_j} \mathbin{||^\Gamma} v_j L_{v_j} \subseteq K \mathbin{||^\Gamma} L$. □

A similar statement can be proven for infinite words.

Lemma 6.4.14. *Let $K, L \subseteq \Delta^\infty$ be limit closed, let Γ be an alphabet, and let $w \in ((\Delta \setminus \Gamma)^* \Gamma)^\omega$. Then*

if $\mathit{pref}(w) \subseteq \mathit{pref}(K) \mathbin{||^\Gamma} \mathit{pref}(L)$, then $w \in K \mathbin{||^\Gamma} L$.

Proof. Let $\mathrm{pref}(w) \subseteq \mathrm{pref}(K) \mathbin{||^\Gamma} \mathrm{pref}(L)$. Let $w_1, w_2, \ldots \in (\Delta \setminus \Gamma)^*$ and $x_1, x_2, \ldots \in \Gamma$ be such that $w = w_1 x_1 w_2 x_2 \cdots$.

Since $\mathrm{pref}(w) \subseteq \mathrm{pref}(K) \mathbin{||^\Gamma} \mathrm{pref}(L)$ we know that for all $n \geq 1$ there exists a sequence $\rho = (u_1, v_1, u_2, v_2, \ldots, u_n, v_n)$, with $u_i, v_i \in (\Delta \setminus \Gamma)^*$ for all $i \in [n]$, and such that $u_1 x_1 u_2 x_2 \cdots u_n x_n \in \mathrm{pref}(K)$, $v_1 x_1 v_2 x_2 \cdots v_n x_n \in \mathrm{pref}(L)$, and $w_i \in (u_i \mathbin{||} v_i)$ for all $i \in [n]$. That is, $w_1 x_1 w_2 x_2 \cdots w_n x_n \in$

$(u_1||v_1)x_1(u_2||v_2)x_2 \cdots (u_n||v_n)x_n = u_1x_1u_2x_2 \cdots u_nx_n||^{\Gamma} v_1x_1v_2x_2 \cdots v_nx_n$. We will refer to $w_1x_1w_2x_2 \cdots w_nx_n$ as $w(n)$ and to ρ as a $(K \, ||^{\Gamma} \, L)$-deco of $w(n)$.

We say that a $(K \, ||^{\Gamma} \, L)$-deco $\rho = (u_1, v_1, u_2, v_2, \ldots, u_n, v_n)$ of $w(n)$ directly precedes a $(K \, ||^{\Gamma} \, L)$-deco ρ' of $w(n+1)$ if $\rho' = (u_1, v_1, u_2, v_2, \ldots, u_n, v_n, u_{n+1}, v_{n+1})$. We furthermore add a trivial ρ_λ which by definition directly precedes every $(K \, ||^{\Gamma} \, L)$-deco of $w(1)$.

For $n \geq 1$, let $V_n = \{\rho \mid \rho \text{ is a } (K \, ||^{\Gamma} \, L)\text{-deco of } w(n)\}$ be the set containing every possible $(K \, ||^{\Gamma} \, L)$-deco of $w(n)$. Let $V_0 = \{\rho_\lambda\}$. Note that each V_n is finite, for $n \geq 0$, and that $V_n \cap V_{n'} = \varnothing$, for all $n > n' \geq 0$. Furthermore, let $E = \{(\rho, \rho') \mid \rho \text{ directly precedes } \rho'\}$. Thus $E \subseteq \bigcup_{n \geq 1}(V_{n-1} \times V_n)$. Note that $G = (\bigcup_{n \geq 0} V_n, E)$ is a directed acyclic graph. In fact, G is an infinite finitely-branching rooted tree with root ρ_λ. This can be seen as follows. Except for ρ_λ, every vertex $\rho = (u_1, v_1, u_2, v_2, \ldots, u_{k+1}, v_{k+1})$ has exactly one incoming edge, viz. from ρ_λ if $k = 0$ and from $(u_1, v_1, u_2, v_2, \ldots, u_k, v_k)$ if $k \geq 1$. Note that this $(u_1, v_1, u_2, v_2, \ldots, u_k, v_k)$ is indeed a $(K \, ||^{\Gamma} \, L)$-deco of $w(k)$. Since each w_i, with $i \in [n]$, is a finite word, every vertex moreover has a finite number of outgoing edges. Finally, the graph is infinite since it has at least one distinct vertex for every prefix $w(n)$ of w.

We can thus use König's Lemma to conclude that there exists an infinite path π through G, starting in the root ρ_λ. Let $\pi = (\rho_\lambda, \rho_1, \rho_2, \ldots)$, with $\rho_n = (u_1, v_1, u_2, v_2, \ldots, u_n, v_n)$ for all $n \geq 1$. Then by definition $u_1x_1u_2x_2 \cdots u_nx_n \in \mathrm{pref}(K)$ and $v_1x_1v_2x_2 \cdots v_nx_n \in \mathrm{pref}(L)$. Since K and L are limit closed this implies that $u = \lim_{n \to \infty} u_1x_1u_2x_2 \cdots u_nx_n \in K$ and $v = \lim_{n \to \infty} v_1x_1v_2x_2 \cdots v_nx_n \in L$. By the definition of the $(K \, ||^{\Gamma} \, L)$-deco of $w(n)$ we thus obtain that $w = w_1x_1w_2x_2 \cdots \in (u_1 \, ||| \, v_1)x_1(u_2 \, ||| \, v_2)x_2 \cdots = u \, ||^{\Gamma} \, v$. Hence $w \in K \, ||^{\Gamma} \, L$. $\square$

The preceding two lemmata allow us to express — as we did for the shuffle in Theorem 6.3.49(2) — the S-shuffle of possibly infinite words in terms of the S-shuffle of finite prefixes of those possibly infinite words.

Theorem 6.4.15. *Let $K, L \subseteq \Delta^\infty$ be limit closed, let $w \in \Delta^\omega$, and let Γ be an alphabet. Then*

$$w \in K \, ||^{\Gamma} \, L \textit{ if and only if } \mathit{pref}(w) \subseteq \mathit{pref}(K) \, ||^{\Gamma} \, \mathit{pref}(L).$$

Proof. (If) Let $\mathrm{pref}(w) \subseteq \mathrm{pref}(K) \, ||^{\Gamma} \, \mathrm{pref}(L)$. Then by Definition 6.4.1 and Lemmata 6.4.13 and 6.4.14 it follows that $w \in K \, ||^{\Gamma} \, L$.

(Only if) Let $w \in K \, ||^{\Gamma} \, L$. Then according to Definition 6.4.1 one of the following two cases holds.
Either $w = (u_1 \, || \, v_1)x_1(u_2 \, || \, v_2)x_2 \cdots x_{n-1}(u_n \, || \, v_n)$ for some $n \geq 1$, $u_i, v_i \in$

$(\Delta \setminus \Gamma)^*$ for all $i \in [n-1]$, $u_n, v_n \in (\Delta \setminus \Gamma)^\infty$, and $x_i \in \Gamma$ for all $i \in [n-1]$, and such that $u = u_1 x_1 u_2 x_2 \cdots x_n u_{n+1}$ and $v = v_1 x_1 v_2 x_2 \cdots x_n v_{n+1}$.
Or else $w = (u_1 \mid\mid v_1) x_1 (u_2 \mid\mid v_2) x_2 \cdots$ for some $n \geq 1$, $u_i, v_i \in (\Delta \setminus \Gamma)^*$ for all $i \geq 1$, and $x_i \in \Gamma$ for all $i \geq 1$, and such that $u = u_1 x_1 u_2 x_2 \cdots x_n u_{n+1}$ and $v = v_1 x_1 v_2 x_2 \cdots x_n v_{n+1}$.

Consequently we consider a prefix $y \in \mathrm{pref}(w)$. Then in both cases $y = (u_1 \mid\mid v_1) x_1 (u_2 \mid\mid v_2) x_2 \cdots x_{k-1} x$ for some $k \geq 1$ and $x \in \mathrm{pref}(u_k \mid\mid v_k)$. Immediately from Definition 6.4.1 and Theorem 6.3.21(1) it then follows that $y \in u_1 x_1 u_2 x_2 \ \cdots \ u_{k-1} x_{k-1} \mathrm{pref}(u_k) \ \mid\mid^\Gamma \ v_1 x_1 v_2 x_2 \ \cdots \ v_{k-1} x_{k-1} \mathrm{pref}(v_k) \subseteq \mathrm{pref}(u) \mid\mid^\Gamma \mathrm{pref}(v)$. Hence $y \in \mathrm{pref}(K) \mid\mid^\Gamma \mathrm{pref}(L)$. □

Since all singleton languages are limit closed, we immediately obtain the following result.

Theorem 6.4.16. *Let $u, v \in \Delta^\infty$, let $w \in \Delta^\omega$, and let Γ be an alphabet. Then*

$$w \in u \mid\mid^\Gamma v \text{ if and only if } \mathit{pref}(w) \subseteq \mathit{pref}(u) \mid\mid^\Gamma \mathit{pref}(v). \qquad \square$$

6.4.3 Commutativity and Associativity

In order to use the (fair) synchronized shuffles in the context of team automata, it is important to establish certain commutativity and associativity properties.

The (fair) S-shuffle is defined on the basis of the (fair) shuffle, which is commutative. Hence the commutativity of the (fair) S-shuffle is a direct consequence of the commutativity of the (fair) shuffle, as stated in Theorem 6.3.8.

Theorem 6.4.17. *Let $u, v \in \Delta^\infty$ and let Γ be an alphabet. Then*

(1) $u \mid\mid\mid^\Gamma v = v \mid\mid\mid^\Gamma u$ and $u \mid\mid^\Gamma v = v \mid\mid^\Gamma u$, and

(2) $L_1 \mid\mid\mid^\Gamma L_2 = L_2 \mid\mid\mid^\Gamma L_1$ and $L_1 \mid\mid^\Gamma L_2 = L_2 \mid\mid^\Gamma L_1$. □

Recall that both rS-shuffles and fS-shuffles are defined in terms of S-shuffles. Consequently, also these synchronized shuffles may be considered commutative in the following sense.

Corollary 6.4.18. *Let $u, v \in \Delta^\infty$, let $L_1, L_2 \subseteq \Delta^\infty$, and let Γ be an alphabet. Then*

(1) $u \ {}_{\Delta_1}\underline{\mid\mid\mid}^\Gamma_{\Delta_2} \ v = v \ {}_{\Delta_2}\underline{\mid\mid\mid}^\Gamma_{\Delta_1} \ u$ and $u \ {}_{\Delta_1}\underline{\mid\mid}^\Gamma_{\Delta_2} \ v = v \ {}_{\Delta_2}\underline{\mid\mid}^\Gamma_{\Delta_1} \ u$, and

(2) $L_1 \ {}_{\Delta_1}\underline{\mid\mid\mid}^\Gamma_{\Delta_2} \ L_2 = L_2 \ {}_{\Delta_2}\underline{\mid\mid\mid}^\Gamma_{\Delta_1} \ L_1$ and $L_1 \ {}_{\Delta_1}\underline{\mid\mid}^\Gamma_{\Delta_2} \ L_2 = L_2 \ {}_{\Delta_2}\underline{\mid\mid}^\Gamma_{\Delta_1} \ L_1$. □

Corollary 6.4.19. *Let $u, v \in \Delta^\infty$ and let $L_1, L_2 \subseteq \Delta^\infty$. Then*

(1) $u\ {}_{\Delta_1}\underline{|||}_{\Delta_2}\ v = v\ {}_{\Delta_2}\underline{|||}_{\Delta_1}\ u$ and $u\ {}_{\Delta_1}\underline{||}_{\Delta_2}\ v = v\ {}_{\Delta_2}\underline{||}_{\Delta_1}\ u$, and

(2) $L_1\ {}_{\Delta_1}\underline{|||}_{\Delta_2}\ L_2 = L_2\ {}_{\Delta_2}\underline{|||}_{\Delta_1}\ L_1$ and $L_1\ {}_{\Delta_1}\underline{||}_{\Delta_2}\ L_2 = L_2\ {}_{\Delta_2}\underline{||}_{\Delta_1}\ L_1$. □

It remains to prove that also in case of synchronized shuffles a notion of associativity can be upheld. In case of S-shuffles, associativity is easily understood. S-shuffling is associative because $\{u\} ||^\Gamma (v ||^\Gamma w)$ equals $(u ||^\Gamma v) ||^\Gamma \{w\}$, for words u, v, and w, and an alphabet Γ, and likewise for the fair case. To prove this statement we use the associativity of (fair) shuffling.

Theorem 6.4.20. *Let $u, v, w \in \Delta^\infty$ and let Γ be an alphabet. Then*

(1) $\{u\} |||^\Gamma (v |||^\Gamma w) = (u |||^\Gamma v) |||^\Gamma \{w\}$ and

(2) $\{u\} ||^\Gamma (v ||^\Gamma w) = (u ||^\Gamma v) ||^\Gamma \{w\}$.

Proof. (1) Let $x \in \{u\} |||^\Gamma (v |||^\Gamma w)$. Then by Lemma 6.4.11, $\mathrm{pres}_\Gamma(x) = \mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v) = \mathrm{pres}_\Gamma(w)$. Now let $y = \mathrm{pres}_\Gamma(x)$. We distinguish two cases.

First consider that $y \in \Gamma^*$. Then there exists an $n \geq 0$ such that $y = y_1 y_2 \cdots y_n$ with $y_i \in \Gamma$, for all $i \in [n]$. Consequently there exist $x_1, x_2, \ldots, x_n, u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n \in \Gamma^*$ and x_{n+1}, $u_{n+1}, v_{n+1}, w_{n+1} \in \Gamma^\infty$ such that $x = x_1 y_1 x_2 y_2 \cdots x_n y_n x_{n+1}$, $u = u_1 y_1 u_2 y_2 \cdots u_n y_n u_{n+1}$, $v = v_1 y_1 v_2 y_2 \cdots v_n y_n v_{n+1}$, and $w = w_1 y_1 w_2 y_2 \cdots w_n y_n w_{n+1}$. By Definition 6.4.1, $x_i \in \{u_i\} || (v_i || w_i)$, for all $i \in [n]$, and $x_{n+1} \in \{u_{n+1}\} ||| (v_{n+1} ||| w_{n+1})$. Now by Theorem 6.3.51(1), $\{u_i\} || (v_i || w_i) = (u_i || v_i) || \{w_i\}$, for all $i \in [n]$, and according to Theorem 6.3.32(1), $\{u_{n+1}\} ||| (v_{n+1} ||| w_{n+1}) = (u_{n+1} ||| v_{n+1}) ||| \{w_{n+1}\}$. This implies, again by Definition 6.4.1, that $x \in (u |||^\Gamma v) |||^\Gamma \{w\}$.

Secondly, the case that $y \in \Gamma^\infty$ is analogous.

(2) Analogous. □

Theorem 6.4.21. *Let $L_1, L_2, L_3 \subseteq \Delta^\infty$ and let Γ be an alphabet. Then*

(1) $L_1 |||^\Gamma (L_2 |||^\Gamma L_3) = (L_1 |||^\Gamma L_2) |||^\Gamma L_3$ and

(2) $L_1 ||^\Gamma (L_2 ||^\Gamma L_3) = (L_1 ||^\Gamma L_2) ||^\Gamma L_3$.

Proof. Analogous to the proof of Theorem 6.3.32(2). □

The statements of the preceding two theorems do not hold when the synchronization alphabet Γ may vary. Given $w_1, w_2, w_3 \in \Delta^*$ and two distinct alphabets Γ and Γ', e.g., $(w_1 ||^\Gamma w_2) ||^{\Gamma'} w_3$ in general does not equal $w_1 ||^\Gamma (w_2 ||^{\Gamma'} w_3)$. This is shown in the following example.

Example 6.4.22. Let $L_1 = \{a\}$, let $L_2 = \{a,b\}$, and let $L_3 = \{ab\}$. Then $(L_1 \shuffle^{\{a\}} L_2) \shuffle^{\{b\}} L_3 = \{a\} \shuffle^{\{b\}} \{ab\} = \varnothing$, whereas $L_1 \shuffle^{\{a\}} (L_2 \shuffle^{\{b\}} L_3) = \{a\} \shuffle^{\{a\}} \{ab\} = \{ab\}$. $\square$

It is worthwhile to notice here that the synchronized shuffle as studied in [DeS84] and [LR99] is not associative, as is noted in [LR99] and shown in the following example. Recall that the 'produit de mixage' or synchonized shuffle of $L_1, L_2 \subseteq \Delta^\infty$ is defined as $L_1 \shuffle^{\mathrm{alph}(L_1)\cap\mathrm{alph}(L_2)} L_2$, where $\mathrm{alph}(L)$ — for an alphabet L — is defined as $\mathrm{alph}(L) = \bigcup_{w\in L}\mathrm{alph}(w)$.

Example 6.4.23. (Example 6.4.22 continued) Now $L_1 \shuffle^{\mathrm{alph}(L_1)\cap\mathrm{alph}(L_2)} L_2 = \{a\} \shuffle^{\{a\}} \{a,b\} = \{a\}$ and thus $\{a\} \shuffle^{\mathrm{alph}(\{a\})\cap\mathrm{alph}(L_3)} L_3 = \{a\} \shuffle^{\{a\}} \{ab\} = \{ab\}$, while on the other hand $L_2 \shuffle^{\mathrm{alph}(L_2)\cap\mathrm{alph}(L_3)} L_3 = \{a,b\} \shuffle^{\{a,b\}} \{ab\} = \varnothing$ and thus $L_1 \shuffle^{\mathrm{alph}(L_1)\cap\mathrm{alph}(\{ab\})} \varnothing = \{a\} \shuffle^{\{a\}} \varnothing = \varnothing$. $\square$

In [vdS85] it is noted that the weave operation studied there is on purpose not defined as the synchronized shuffle operation of [DeS84] and [LR99] because in that case it would no longer have been associative.

Contrary to the case of the S-shuffle, the synchronization alphabet of an fS-shuffle or an rS-shuffle depends on the alphabets involved. Hence it is not immediately clear how associativity should be formalized. A natural approach would be to consider fS-shuffling associative if $\{u\}\ {}_{\Delta_1}\underline{\shuffle}_{\Delta_2\cup\Delta_3}\ (v\ {}_{\Delta_2}\underline{\shuffle}_{\Delta_3}\ w)$ equals $(u\ {}_{\Delta_1}\underline{\shuffle}_{\Delta_2}\ v)\ {}_{\Delta_1\cup\Delta_2}\underline{\shuffle}_{\Delta_3}\ \{w\}$ for all words $u \in \Delta_1^\infty$, $v \in \Delta_2^\infty$, and $w \in \Delta_3^\infty$, and similarly rS-shuffling and the fair cases.

We now present an example to illustrate this idea.

Example 6.4.24. (Example 6.4.4 continued) Recall that we have set $\Delta_1 = \{a,b,c\}$, $\Delta_2 = \{c,d\}$, $u = abc \in \Delta_1^*$, and $v = cd \in \Delta_2^*$. Now we let $\Delta_3 = \{b,c,e\}$ and we let $w = bce \in \Delta_3^*$. Then it follows immediately that $\{u\}\ {}_{\Delta_1}\underline{\shuffle}_{\Delta_2\cup\Delta_3}\ (v\ {}_{\Delta_2}\underline{\shuffle}_{\Delta_3}\ w) = \{abc\}\ {}_{\{a,b,c\}}\underline{\shuffle}_{\{b,c,d,e\}}\ (cd\ {}_{\{c,d\}}\underline{\shuffle}_{\{b,c,e\}}\ bce) = \{abc\}\ {}_{\{a,b,c\}}\underline{\shuffle}_{\{b,c,d,e\}}\ \{bcde, bced\} = \{abcde, abced\} = abcd\ {}_{\{a,b,c,d\}}\underline{\shuffle}_{\{b,c,e\}}\ bce = (abc\ {}_{\{a,b,c\}}\underline{\shuffle}_{\{c,d\}}\ cd)\ {}_{\{a,b,c,d\}}\underline{\shuffle}_{\{b,c,e\}}\ \{bce\} = (u\ {}_{\Delta_1}\underline{\shuffle}_{\Delta_2}\ v)\ {}_{\Delta_1\cup\Delta_2}\underline{\shuffle}_{\Delta_3}\ \{w\}$.

Next we let $\Gamma = \{b,c\}$. Consequently, it follows immediately that $\{u\}\ {}_{\Delta_1}\underline{\shuffle}^{\Gamma}_{\Delta_2\cup\Delta_3}\ (v\ {}_{\Delta_2}\underline{\shuffle}^{\Gamma}_{\Delta_3}\ w) = \{abc\}\ {}_{\{a,b,c\}}\underline{\shuffle}^{\{b,c\}}_{\{b,c,d,e\}}\ (cd\ {}_{\{c,d\}}\underline{\shuffle}^{\{b,c\}}_{\{b,c,e\}}\ bce) = \{abc\}\ {}_{\{a,b,c\}}\underline{\shuffle}^{\{b,c\}}_{\{b,c,d,e\}}\ \{bcde, bced\} = \{abcde, abced\} = abcd\ {}_{\{a,b,c,d\}}\underline{\shuffle}^{\{b,c\}}_{\{b,c,e\}}\ bce = (abc\ {}_{\{a,b,c\}}\underline{\shuffle}^{\{b,c\}}_{\{c,d\}}\ cd)\ {}_{\{a,b,c,d\}}\underline{\shuffle}^{\{b,c\}}_{\{b,c,e\}}\ \{bce\} = (u\ {}_{\Delta_1}\underline{\shuffle}^{\Gamma}_{\Delta_2}\ v)\ {}_{\Delta_1\cup\Delta_2}\underline{\shuffle}^{\Gamma}_{\Delta_3}\ \{w\}$. $\square$

This example dealt with finite words and hence fair fS-shuffles and fair rS-shuffles. Before turning to the general case we now prove that indeed fair fS-shuffling and fair rS-shuffling are associative in the sense just discussed. The following characterization of the fair shuffles of two words over disjoint alphabets in terms of preserving homomorphisms turns out to be very useful.

We give here a full direct proof, but the statement can also be proven by modification of Theorem 6.3.29 and its proof (using $\mathrm{pres}_{\Delta_1}^{-1}$ and $\mathrm{pres}_{\Delta_2}^{-1}$ instead of the inverse homomorphisms $\varphi_{i,\{j\}}^{-1}$ and $\varphi_{j,\{i\}}^{-1}$).

Lemma 6.4.25. *Let $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$ be such that $\Delta_1 \cap \Delta_2 = \varnothing$. Then*

$$u \mid\mid\mid v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \mathit{pres}_{\Delta_1}(w) = u,\ \mathit{pres}_{\Delta_2}(w) = v\}.$$

Proof. ($\subseteq$) Let $w \in u \mid\mid\mid v$. Since $\Delta_1 \cap \Delta_2 = \varnothing$ it follows immediately by Lemma 6.3.7(1) that $\mathrm{pres}_{\Delta_1}(w) = u$ and $\mathrm{pres}_{\Delta_2}(w) = v$.

($\supseteq$) Let $w \in (\Delta_1 \cup \Delta_2)^\infty$ be such that $\mathrm{pres}_{\Delta_1}(w) = u$ and $\mathrm{pres}_{\Delta_2}(w) = v$. We distinguish three cases.

First consider that $u \in \Delta_1^*$. Since $\mathrm{pres}_{\Delta_1}(w) = u$ there exist an $n \geq 0$ and $a_1, a_2, \ldots, a_n \in \Delta_1$ such that $u = a_1 a_2 \cdots a_n$ and $w = \alpha_0 a_1 \alpha_1 a_2 \cdots a_n \alpha_n$, where $\alpha_0, \alpha_1, \ldots, \alpha_{n-1} \in \Delta_2^*$ and $\alpha_n \in \Delta_2^\infty$. Since $\mathrm{pres}_{\Delta_2}(w) = v$ and $\Delta_1 \cap \Delta_2 = \varnothing$, we have $v = \alpha_0 \alpha_1 \cdots \alpha_n$. Now let $\alpha_n = \lim_{m \to \infty} \gamma_1 \gamma_2 \cdots \gamma_m$ with $\gamma_i \in \Delta_2^*$, for all $i \geq 1$. Hence $w = \alpha_0 a_1 \alpha_1 a_2 \cdots \alpha_{n-1} a_n \gamma_1 \lambda \gamma_2 \lambda \cdots$ with $u = a_1 a_2 \cdots a_n$ and $v = \alpha_1 \alpha_2 \cdots \alpha_{n-1} \gamma_1 \gamma_2 \cdots$ and thus, again by Lemma 6.3.7(1), $w \in u \mid\mid\mid v$.

The case that $v \in \Delta_2^*$ is analogous.

Finally, consider that $u \in \Delta_1^\omega$ and $v \in \Delta_2^\omega$. Hence $w \in (\Delta_1 \cup \Delta_2)^\omega$. Let $w = c_1 c_2 \cdots = \lim_{n \to \infty} c_1 c_2 \cdots c_n$ with $c_i \in \Delta_1 \cup \Delta_2$, for all $i \geq 1$. By the definition of homomorphisms on infinite words, $\mathrm{pres}_{\Delta_1}(w) = \lim_{n \to \infty} \mathrm{pres}_{\Delta_1}(c_1 c_2 \cdots c_n) = u$ and $\mathrm{pres}_{\Delta_2}(w) = \lim_{n \to \infty} \mathrm{pres}_{\Delta_2}(c_1 c_2 \cdots c_n) = v$. Now denote $\mathrm{pres}_{\Delta_1}(c_1 c_2 \cdots c_n)$ by u_n and $\mathrm{pres}_{\Delta_2}(c_1 c_2 \cdots c_n)$ by v_n. From the first two cases it then follows that for all $n \geq 1$, $c_1 c_2 \cdots c_n \in u_n \mid\mid\mid v_n$. Hence $\mathrm{pref}(w) \subseteq \mathrm{pref}(u) \mid\mid\mid \mathrm{pref}(v)$, which implies that $w \in u \mid\mid v$ by Corollary 6.3.48. Since $\Delta_1 \cap \Delta_2 = \varnothing$ and u and v are both infinite words, w satisfies subcase (c) of case (4) of Definition 6.3.1 and thus $w \in u \mid\mid\mid v$. $\square$

This result implies that also the fair S-shuffles and the fair fS-shuffles can be described in terms of preserving homomorphisms, provided that there is no confusion about the non-synchronizing symbols.

Theorem 6.4.26. *Let Γ be an alphabet and let $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$ be such that $(\Delta_1 \setminus \Gamma) \cap (\Delta_2 \setminus \Gamma) = \varnothing$. Then*

$$u \mid\mid\mid^\Gamma v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \mathit{pres}_\Gamma(w) = \mathit{pres}_\Gamma(u) = \mathit{pres}_\Gamma(v),\ \mathit{pres}_{\Delta_1}(w) = u,\ \mathit{pres}_{\Delta_2}(w) = v\}.$$

Proof. ($\subseteq$) Let $w \in u \mid\mid\mid^\Gamma v$. As by Lemma 6.4.11(1), $\mathrm{pres}_\Gamma(w) = \mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v)$, we only have to prove that $\mathrm{pres}_{\Delta_1}(w) = u$ and $\mathrm{pres}_{\Delta_2}(w) = v$. According to Definition 6.4.1 we can distinguish two cases.

First consider that $w = w_1 y_1 w_2 y_2 \cdots y_n w_{n+1}$, where for some $n \geq 1$, $w_1, w_2, \ldots, w_n \in ((\Delta_1 \cup \Delta_2) \setminus \Gamma)^*$, $w_{n+1} \in ((\Delta_1 \cup \Delta_2) \setminus \Gamma)^\infty$, $y_1, y_2, \ldots, y_n \in \Gamma$, $u = u_1 y_1 u_2 y_2 \cdots y_n u_{n+1}$, with $u_1, u_2, \ldots, u_n \in (\Delta_1 \setminus \Gamma)^*$ and $u_{n+1} \in (\Delta_1 \setminus \Gamma)^\infty$, and $v = v_1 y_1 v_2 y_2 \cdots y_n v_{n+1}$, with $v_1, v_2, \ldots, v_n \in (\Delta_2 \setminus \Gamma)^*$ and $v_{n+1} \in (\Delta_2 \setminus \Gamma)^\infty$, are such that for all $i \in [n+1]$, $w_i \in u_i \,|||\, v_i$. Since $(\Delta_1 \setminus \Gamma) \cap (\Delta_2 \setminus \Gamma) = \varnothing$, it follows from Lemma 6.4.25 that for all $i \in [n+1]$, $\mathrm{pres}_{\Delta_1}(w_i) = u_i$ and $\mathrm{pres}_{\Delta_2}(w_i) = v_i$. Hence $\mathrm{pres}_{\Delta_1}(w) = \mathrm{pres}_{\Delta_1}(w_1)\,\mathrm{pres}_{\Delta_1}(y_1)\,\mathrm{pres}_{\Delta_1}(w_2)\,\mathrm{pres}_{\Delta_1}(y_2) \cdots \mathrm{pres}_{\Delta_1}(y_n)\,\mathrm{pres}_{\Delta_1}(w_{n+1}) = u_1 y_1 u_2 y_2 \cdots y_n u_{n+1} = u$ and, analogously, $\mathrm{pres}_{\Delta_2}(w) = v_1 y_1 v_2 y_2 \cdots y_n v_{n+1} = v$.

Secondly, consider that $w = w_1 y_1 w_2 y_2 \cdots$, where $w_1, w_2, \ldots \in ((\Delta_1 \cup \Delta_2) \setminus \Gamma)^*$, $y_1, y_2, \ldots \in \Gamma$, $u = u_1 y_1 u_2 y_2 \cdots$, with $u_1, u_2, \ldots \in (\Delta_1 \setminus \Gamma)^*$, and $v = v_1 y_1 v_2 y_2 \cdots$, with $v_1, v_2, \ldots \in (\Delta_2 \setminus \Gamma)^*$, are such that for all $i \geq 1$, $w_i \in u_i \,|||\, v_i$. Since $(\Delta_1 \setminus \Gamma) \cap (\Delta_2 \setminus \Gamma) = \varnothing$, Lemma 6.4.25 implies that for all $i \geq 1$, $\mathrm{pres}_{\Delta_1}(w_i) = u_i$ and $\mathrm{pres}_{\Delta_2}(w_i) = v_i$. Hence, by the definition of homomorphisms on infinite words, $\mathrm{pres}_{\Delta_1}(w) = u_1 y_1 u_2 y_2 \cdots = u$ and $\mathrm{pres}_{\Delta_2}(w) = v_1 y_1 v_2 y_2 \cdots = v$.

($\supseteq$) Let $w \in (\Delta_1 \cup \Delta_2)^\infty$ be such that $\mathrm{pres}_\Gamma(w) = \mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v)$, $\mathrm{pres}_{\Delta_1}(w) = u$, and $\mathrm{pres}_{\Delta_2}(w) = v$. Observe that $(\Delta_1 \setminus \Gamma) \cap (\Delta_2 \setminus \Gamma) = \varnothing$ implies that $\Delta_1 \cap \Delta_2 \subseteq \Gamma$. Hence, by Lemma 6.4.7(2), $u \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; v = u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v$. Moreover, since $\mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v)$, we have $w \in u \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; v$ if and only if $w \in u \,|||^{\Gamma}\, v$. Thus it suffices to prove that $w \in u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v$. We distinguish two cases.

First consider that $\mathrm{pres}_{\Delta_1 \cap \Delta_2}(w) \in (\Delta_1 \cup \Delta_2)^*$. Then there exists an $n \geq 1$ such that $w = w_1 y_1 w_2 y_2 \cdots y_n w_{n+1}$, where for all $i \in [n]$, $w_i \in ((\Delta_1 \setminus \Delta_2) \cup (\Delta_2 \setminus \Delta_1))^*$ and $y_i \in \Delta_1 \cap \Delta_2$, and $w_{n+1} \in ((\Delta_1 \setminus \Delta_2) \cup (\Delta_2 \setminus \Delta_1))^\infty$. Moreover, $\mathrm{pres}_{\Delta_1}(w) = \mathrm{pres}_{\Delta_1}(w_1) y_1 \mathrm{pres}_{\Delta_1}(w_2) y_2 \cdots y_n \mathrm{pres}_{\Delta_1}(w_{n+1}) = u$ and $\mathrm{pres}_{\Delta_2}(w) = \mathrm{pres}_{\Delta_2}(w_1) y_1 \mathrm{pres}_{\Delta_2}(w_2) y_2 \cdots y_n \mathrm{pres}_{\Delta_2}(w_{n+1}) = v$. Hence $u = u_1 y_1 u_2 y_2 \cdots y_n u_{n+1}$, with $u_i = \mathrm{pres}_{\Delta_1 \setminus \Delta_2}(w_i)$, for all $i \in [n+1]$, and $v = v_1 y_1 v_2 y_2 \cdots y_n v_{n+1}$, with $v_i = \mathrm{pres}_{\Delta_2 \setminus \Delta_1}(w_i)$, for all $i \in [n+1]$. Since for all $i \in [n+1]$, $u_i \in (\Delta_1 \setminus \Delta_2)^\infty$, $v_i \in (\Delta_2 \setminus \Delta_1)^\infty$, and $(\Delta_1 \setminus \Delta_2) \cap (\Delta_2 \setminus \Delta_1) = \varnothing$, Lemma 6.4.25 implies that for all $i \in [n]$, $w_i \in u_i \,|||\, v_i = u_i \,||\, v_i$, and $w_{n+1} \in u_{n+1} \,|||\, v_{n+1} \subseteq u_{n+1} \,||\, v_{n+1}$. Definition 6.4.1(1) now implies that $w \in u \,|||^{\Delta_1 \cap \Delta_2}\, v$, which by Definition 6.4.3 means that $w \in u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v$.

Next consider that $\mathrm{pres}_{\Delta_1 \cap \Delta_2}(w) \in (\Delta_1 \cup \Delta_2)^\omega$. Then $w = w_1 y_1 w_2 y_2 \cdots$, where for all $i \geq 1$, $w_i \in ((\Delta_1 \setminus \Delta_2) \cup (\Delta_2 \setminus \Delta_1))^*$ and $y_i \in \Delta_1 \cap \Delta_2$. Moreover, $\mathrm{pres}_{\Delta_1}(w) = \mathrm{pres}_{\Delta_1}(w_1) y_1 \mathrm{pres}_{\Delta_1}(w_2) y_2 \cdots = u$ and $\mathrm{pres}_{\Delta_2}(w) = \mathrm{pres}_{\Delta_2}(w_1) y_1 \mathrm{pres}_{\Delta_2}(w_2) y_2 \cdots = v$. Hence $u = u_1 y_1 u_2 y_2 \cdots$, with $u_i = \mathrm{pres}_{\Delta_1 \setminus \Delta_2}(w_i)$, for all $i \geq 1$, and $v = v_1 y_1 v_2 y_2 \cdots$, with $v_i = \mathrm{pres}_{\Delta_2 \setminus \Delta_1}(w_i)$, for all $i \geq 1$. Since for all $i \geq 1$, $u_i \in (\Delta_1 \setminus \Delta_2)^*$, $v_i \in (\Delta_2 \setminus \Delta_1)^*$, and $(\Delta_1 \setminus \Delta_2) \cap (\Delta_2 \setminus \Delta_1) = \varnothing$, Lemma 6.4.25 implies that for all $i \geq 1$,

$w_i \in u_i \mathbin{|||} v_i = u_i \mathbin{||} v_i$. Definition 6.4.1(2) now implies that $w \in u \mathbin{|||}^{\Delta_1 \cap \Delta_2} v$, which by Definition 6.4.3 means that $w \in u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v$. □

Finally, we obtain from this result a characterization of fair fS-shuffling.

Corollary 6.4.27. *Let $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$. Then*

$$u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid pres_{\Delta_1}(w) = u,\ pres_{\Delta_2}(w) = v\}.$$

Proof. By Definition 6.4.3, $u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v = u \mathbin{|||}^{\Delta_1 \cap \Delta_2} v$ and $(\Delta_1 \setminus (\Delta_1 \cap \Delta_2)) \cap (\Delta_2 \setminus (\Delta_1 \cap \Delta_2)) = \varnothing$. Moreover, if $\mathrm{pres}_{\Delta_1}(w) = u$ and $\mathrm{pres}_{\Delta_2}(w) = v$, then $\mathrm{pres}_{\Delta_1 \cap \Delta_2}(w) = \mathrm{pres}_{\Delta_1}(\mathrm{pres}_{\Delta_2}(\mathrm{pres}_{\Delta_2}(w))) = \mathrm{pres}_{\Delta_1}(\mathrm{pres}_{\Delta_2}(v)) = \mathrm{pres}_{\Delta_1 \cap \Delta_2}(v)$. Similarly, $\mathrm{pres}_{\Delta_1 \cap \Delta_2}(w) = \mathrm{pres}_{\Delta_1 \cap \Delta_2}(u)$. Hence, by Theorem 6.4.26, $u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v = u \mathbin{|||}^{\Delta_1 \cap \Delta_2} v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \mathrm{pres}_{\Delta_1 \cap \Delta_2}(w) = \mathrm{pres}_{\Delta_1 \cap \Delta_2}(u) = \mathrm{pres}_{\Delta_1 \cap \Delta_2}(v),\ \mathrm{pres}_{\Delta_1}(w) = u,\ \mathrm{pres}_{\Delta_2}(w) = v\} = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \mathrm{pres}_{\Delta_1}(w) = u,\ \mathrm{pres}_{\Delta_2}(w) = v\}$. □

Now we can prove the associativity of fair fS-shuffling.

Theorem 6.4.28. *Let $u \in \Delta_1^\infty$, let $v \in \Delta_2^\infty$, and let $w \in \Delta_3^\infty$. Then*

$$\{u\} \;{}_{\Delta_1}\underline{|||}_{\Delta_2 \cup \Delta_3}\; (v \;{}_{\Delta_2}\underline{|||}_{\Delta_3}\; w) = (u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v) \;{}_{\Delta_1 \cup \Delta_2}\underline{|||}_{\Delta_3}\; \{w\}.$$

Proof. ($\subseteq$) Let $x \in \{u\} \;{}_{\Delta_1}\underline{|||}_{\Delta_2 \cup \Delta_3}\; (v \;{}_{\Delta_2}\underline{|||}_{\Delta_3}\; w)$ and let $y \in v \;{}_{\Delta_2}\underline{|||}_{\Delta_3}\; w$ be such that $x \in \;{}_{\Delta_1}\underline{|||}_{\Delta_2 \cup \Delta_3}\; y$. Hence, according to Corollary 6.4.27, $\mathrm{pres}_{\Delta_1}(w) = u$ and $\mathrm{pres}_{\Delta_2 \cup \Delta_3}(w) = y$. Moreover, $\mathrm{pres}_{\Delta_2}(y) = v$ and $\mathrm{pres}_{\Delta_3}(y) = w$. Now let $z = \mathrm{pres}_{\Delta_1 \cup \Delta_2}(x)$. By repeatedly applying Corollary 6.4.27 and by using the properties of preserving homomorphisms, we obtain that $\mathrm{pres}_{\Delta_3}(x) = \mathrm{pres}_{\Delta_3}(\mathrm{pres}_{\Delta_2 \cup \Delta_3}(x)) = \mathrm{pres}_{\Delta_3}(y) = w$, and thus $x \in z \;{}_{\Delta_1 \cup \Delta_2}\underline{|||}_{\Delta_3}\; w$. Furthermore, $\mathrm{pres}_{\Delta_1}(z) = \mathrm{pres}_{\Delta_1}(\mathrm{pres}_{\Delta_1 \cup \Delta_2}(x)) = \mathrm{pres}_{\Delta_1}(x) = u$ and $\mathrm{pres}_{\Delta_2}(z) = \mathrm{pres}_{\Delta_2}(\mathrm{pres}_{\Delta_1 \cup \Delta_2}(x)) = \mathrm{pres}_{\Delta_2}(x) = \mathrm{pres}_{\Delta_2}(\mathrm{pres}_{\Delta_2 \cup \Delta_3}(x)) = \mathrm{pres}_{\Delta_2}(y) = v$. Hence $z \in u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v$ and thus we have proven that $x \in (u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v) \;{}_{\Delta_1 \cup \Delta_2}\underline{|||}_{\Delta_3}\; \{w\}$.

($\supseteq$) By Corollary 6.4.19 and ($\subseteq$) we immediately obtain that $(u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v) \;{}_{\Delta_1 \cup \Delta_2}\underline{|||}_{\Delta_3}\; \{w\} = \{w\} \;{}_{\Delta_3}\underline{|||}_{\Delta_1 \cup \Delta_2} (u \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; v) = \{w\} \;{}_{\Delta_3}\underline{|||}_{\Delta_1 \cup \Delta_2} (v \;{}_{\Delta_2}\underline{|||}_{\Delta_1}\; u) \subseteq (w \;{}_{\Delta_3}\underline{|||}_{\Delta_2}\; v) \;{}_{\Delta_2 \cup \Delta_3}\underline{|||}_{\Delta_1}\; \{u\} = (v \;{}_{\Delta_2}\underline{|||}_{\Delta_3}\; w) \;{}_{\Delta_2 \cup \Delta_3}\underline{|||}_{\Delta_1}\; \{u\} = \{u\} \;{}_{\Delta_1}\underline{|||}_{\Delta_2 \cup \Delta_3} (v \;{}_{\Delta_2}\underline{|||}_{\Delta_3}\; w)$. □

This result can be lifted to languages.

Theorem 6.4.29. *Let $L_1 \subseteq \Delta_1^\infty$, let $L_2 \subseteq \Delta_2^\infty$, and let $L_3 \subseteq \Delta_3^\infty$. Then*

$$L_1 \;{}_{\Delta_1}\underline{|||}_{\Delta_2 \cup \Delta_3}\; (L_2 \;{}_{\Delta_2}\underline{|||}_{\Delta_3}\; L_3) = (L_1 \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; L_2) \;{}_{\Delta_1 \cup \Delta_2}\underline{|||}_{\Delta_3}\; L_3.$$

Proof. Analogous to the proof of Theorem 6.3.32(2). □

The fact that the rS-shuffle is defined in terms of the S-shuffle, together with the associativity of fair fS-shuffling, now allows us to prove that also fair rS-shuffling is associative.

Theorem 6.4.30. *Let $u \in \Delta_1^\infty$, let $v \in \Delta_2^\infty$, let $w \in \Delta_3^\infty$, and let Γ be an alphabet. Then*

$$\{u\} \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2 \cup \Delta_3}\; (v \;{}_{\Delta_2}\underline{|||}^{\Gamma}_{\Delta_3}\; w) = (u \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; v) \;{}_{\Delta_1 \cup \Delta_2}\underline{|||}^{\Gamma}_{\Delta_3}\; \{w\}.$$

Proof. Since fair rS-shuffles are defined in terms of fair S-shuffles, it is clear that it suffices to prove that $\{u\} \;|||^{\Gamma \cap (\Delta_1 \cap (\Delta_2 \cup \Delta_3))}\; (v \;|||^{\Gamma \cap \Delta_2 \cap \Delta_3}\; w) = (u \;|||^{\Gamma \cap \Delta_1 \cap \Delta_2}\; v) \;|||^{\Gamma \cap (\Delta_1 \cup \Delta_2) \cap \Delta_3}\; \{w\}$.

Let $\Delta^\ell = \{a_\ell \mid a \in \Delta\}$, for all $\ell \in \{[123],[12],[13],[23],[1],[2],[3]\}$. Consequently, we consider the homomorphism $\varphi : (\Delta_1 \cup \Delta_2 \cup \Delta_3)^\infty \to (\Delta^{[123]} \cup \Delta^{[12]} \cup \Delta^{[13]} \cup \Delta^{[23]} \cup \Delta^{[1]} \cup \Delta^{[2]} \cup \Delta^{[3]})^\infty$, which we use to label each letter from u, v, and w in a specific way: those letters that appear in Γ and in at least two of the alphabets Δ_1, Δ_2, or Δ_3, are labeled by subscripts indicating all of the alphabets from Δ_1, Δ_2, or Δ_3 that they appear in, while all other letters are labeled by subscripts indicating the unique alphabet from Δ_1, Δ_2, or Δ_3 that they appear in. Formally, φ is defined as follows.

$$\varphi(a) = \begin{cases} a_{[123]} & \text{if } a \in \Gamma \cap \Delta_1 \cap \Delta_2 \cap \Delta_3, \\ a_{[12]} & \text{if } a \in (\Gamma \cap \Delta_1 \cap \Delta_2) \setminus \Delta_3, \\ a_{[13]} & \text{if } a \in (\Gamma \cap \Delta_1 \cap \Delta_3) \setminus \Delta_2, \\ a_{[23]} & \text{if } a \in (\Gamma \cap \Delta_2 \cap \Delta_3) \setminus \Delta_1, \\ a_{[1]} & \text{if } a \in (\Delta_1 \setminus \Gamma) \cup ((\Gamma \cap \Delta_1) \setminus (\Delta_2 \cup \Delta_3)), \\ a_{[2]} & \text{if } a \in (\Delta_2 \setminus \Gamma) \cup ((\Gamma \cap \Delta_2) \setminus (\Delta_1 \cup \Delta_3)), \text{ and} \\ a_{[3]} & \text{if } a \in (\Delta_3 \setminus \Gamma) \cup ((\Gamma \cap \Delta_3) \setminus (\Delta_1 \cup \Delta_2)). \end{cases}$$

Now let $\hat{\Delta}_1 = \Delta^{[123]} \cup \Delta^{[12]} \cup \Delta^{[13]} \cup \Delta^{[1]}$, let $\hat{\Delta}_2 = \Delta^{[123]} \cup \Delta^{[12]} \cup \Delta^{[23]} \cup \Delta^{[2]}$, and let $\hat{\Delta}_3 = \Delta^{[123]} \cup \Delta^{[13]} \cup \Delta^{[23]} \cup \Delta^{[3]}$. Hence $\varphi(u) \in \hat{\Delta}_1^\infty$, $\varphi(v) \in \hat{\Delta}_2^\infty$, and $\varphi(w) \in \hat{\Delta}_3^\infty$. From the way we have labeled the alphabets we obtain that $a \in \Gamma \cap (\Delta_1 \cap (\Delta_2 \cup \Delta_3))$ if and only if $a \in (\Gamma \cap \Delta_1 \cap \Delta_2 \cap \Delta_3) \cup ((\Gamma \cap \Delta_1 \cap \Delta_2) \setminus \Delta_3) \cup ((\Gamma \cap \Delta_1 \cap \Delta_3) \setminus \Delta_2)$ if and only if $\varphi(a) \in \Delta^{[123]} \cup \Delta^{[12]} \cup \Delta^{[13]}$ if and only if $\varphi(a) \in \hat{\Delta}_1 \cap (\hat{\Delta}_2 \cup \hat{\Delta}_3)$ and similarly for the other (potential) synchronization symbols. Since φ is injective, it thus follows that $\{u\} \;|||^{\Gamma \cap (\Delta_1 \cap (\Delta_2 \cup \Delta_3))}\; (v \;|||^{\Gamma \cap \Delta_2 \cap \Delta_3}\; w) = \varphi^{-1}(\varphi(u) \;|||^{\hat{\Delta}_1 \cap (\hat{\Delta}_2 \cup \hat{\Delta}_3)}\; (\varphi(v) \;|||^{\hat{\Delta}_2 \cap \hat{\Delta}_3}\; \varphi(w)))$, which by the associativity of Theorem 6.4.28 is equal to $\varphi^{-1}((\varphi(u) \;|||^{\hat{\Delta}_1 \cap \hat{\Delta}_2}\; \varphi(v)) \;|||^{(\hat{\Delta}_1 \cup \hat{\Delta}_2) \cap \hat{\Delta}_3}\; \varphi(w))$ and this, once again by the labeling of the alphabets, equals $(u \;|||^{\Gamma \cap \Delta_1 \cap \Delta_2}\; v) \;|||^{\Gamma \cap (\Delta_1 \cup \Delta_2) \cap \Delta_3}\; \{w\}$. □

The associativity of fair rS-shuffling can also be proven for languages.

Theorem 6.4.31. *Let* $L_1 \subseteq \Delta_1^\infty$*, let* $L_2 \subseteq \Delta_2^\infty$*, let* $L_3 \subseteq \Delta_3^\infty$*, and let* Γ *be an alphabet. Then*

$$L_1 \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2\cup\Delta_3} \ (L_2 \ {}_{\Delta_2}\underline{|||}^{\Gamma}_{\Delta_3} \ L_3) = (L_1 \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2} \ L_2) \ {}_{\Delta_1\cup\Delta_2}\underline{|||}^{\Gamma}_{\Delta_3} \ L_3.$$

Proof. Analogous to the proof of Theorem 6.3.32(2). □

As was the case for the associativity of S-shuffling, also the statements of the preceding two theorems do not hold when Γ may vary. Given $w_i \in \Delta_i^*$, with $i \in [3]$, and two distinct alphabets Γ and Γ', e.g., in general $(w_1 \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2} \ w_2) \ {}_{\Delta_1\cup\Delta_2}\underline{|||}^{\Gamma'}_{\Delta_3} \ w_3$ does not equal $w_1 \ {}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2\cup\Delta_3} \ (w_2 \ {}_{\Delta_2}\underline{|||}^{\Gamma'}_{\Delta_3} \ w_3)$. This is shown in the following example.

Example 6.4.32. Let $\Delta_1 = \{a\}$, let $\Delta_2 = \{b\}$, and let $\Delta_3 = \{a, b\}$. Then clearly $(a \ {}_{\Delta_1}\underline{|||}^{\{a\}}_{\Delta_2} \ b) \ {}_{\Delta_1\cup\Delta_2}\underline{|||}^{\{b\}}_{\Delta_3} \ \{ab\} = (a \ {}_{\{a\}}\underline{|||}^{\{a\}}_{\{b\}} \ b) \ {}_{\{a,b\}}\underline{|||}^{\{b\}}_{\{a,b\}} \ \{ab\} = \{ab, ba\} \ {}_{\{a,b\}}\underline{||}^{\{b\}}_{\{a,b\}} \ \{ab\} = \{aab, aba\}$, while $\{a\} \ {}_{\Delta_1}\underline{|||}^{\{a\}}_{\Delta_2\cup\Delta_3} \ (b \ {}_{\Delta_2}\underline{|||}^{\{b\}}_{\Delta_3} \ ab) = \{a\} \ {}_{\{a\}}\underline{|||}^{\{a\}}_{\{a,b\}} \ (b \ {}_{\{b\}}\underline{|||}^{\{b\}}_{\{a,b\}} \ ab) = a \ {}_{\{a\}}\underline{|||}^{\{a\}}_{\{a,b\}} \ ab = \{ab\}$. □

In case of "unfair" fS-shuffling (and thus also in case of "unfair" rS-shuffling) the associativity at the level of words which we have established for the fair case (and for S-shuffling) does not hold. As the following example shows, unfair fS-shuffling with an infinite word may lead to the abortion of its finite partner and thus destroy the associativity.

Example 6.4.33. Let $\Delta_1 = \{a, b\}$, let $\Delta_2 = \{b\}$, and let $\Delta_3 = \{c\}$. Then clearly $a \ {}_{\Delta_1}\underline{||}_{\Delta_2} \ b = \varnothing$ and thus $(a \ {}_{\Delta_1}\underline{||}_{\Delta_2} \ b) \ {}_{\Delta_1\cup\Delta_2}\underline{||}_{\Delta_3} \ \{c^\omega\} = \varnothing$. However, $\{a\} \ {}_{\Delta_1}\underline{||}_{\Delta_2\cup\Delta_3} \ (b \ {}_{\Delta_2}\underline{||}_{\Delta_3} \ c^\omega) = \{a\} \ {}_{\{a,b\}}\underline{||}_{\{b,c\}} \ (b \ {}_{\{b\}}\underline{||}_{\{c\}} \ c^\omega) = \{a\} \ {}_{\{a,b\}}\underline{||}_{\{b,c\}} \ (\{c^n b c^\omega \mid n \geq 0\} \cup \{c^\omega\}) = \{c^n a c^\omega \mid n \geq 0\} \cup \{c^\omega\}$. □

At first sight, adding λ to represent possible abortion appears to be a solution. For this example, adding λ indeed solves the problem, as is shown in the following example.

Example 6.4.34. (Example 6.4.33 continued) We show how adding λ to a, b, and c^ω may solve the problem, viz. $(\{\lambda, a\} \ {}_{\Delta_1}\underline{||}_{\Delta_2} \ \{\lambda, b\}) \ {}_{\Delta_1\cup\Delta_2}\underline{||}_{\Delta_3} \ \{\lambda, c^\omega\} = (\{\lambda, a\} \ {}_{\{a,b\}}\underline{||}_{\{b\}} \ \{\lambda, b\}) \ {}_{\{a,b\}}\underline{||}_{\{c\}} \ \{\lambda, c^\omega\} = \{\lambda, a\} \ {}_{\{a,b\}}\underline{||}_{\{c\}} \ \{\lambda, c^\omega\} = \{\lambda, a, c^\omega\} \cup \{c^n a c^\omega \mid n \geq 0\} = \{\lambda, a\} \ {}_{\{a,b\}}\underline{||}_{\{b,c\}} \ (\{\lambda, b, c^\omega\} \cup \{c^n b c^\omega \mid n \geq 0\}) = \{\lambda, a\} \ {}_{\{a,b\}}\underline{||}_{\{b,c\}} \ (\{\lambda, b\} \ {}_{\{b\}}\underline{||}_{\{c\}} \ \{\lambda, c^\omega\}) = \{\lambda, a\} \ {}_{\Delta_1}\underline{||}_{\Delta_2\cup\Delta_3} \ (\{\lambda, b\} \ {}_{\Delta_2}\underline{||}_{\Delta_3} \ \{\lambda, c^\omega\})$. □

In this example the aborted word consists of one symbol only and thus has λ as its only proper prefix, whereas in general infinite words when unfairly shuffled may still tolerate arbitrary prefixes, in which case adding λ is not a solution. This is shown in the following example.

Example 6.4.35. Note $(\{\lambda, a^\omega\} \;{}_{\{a\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b^2\}) \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b\} = (\{\lambda, b^2, a^\omega\} \cup \{a^m b a^n b a^\omega, a^n b a^\omega \mid m, n \geq 0\}) \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b\} = \{\lambda, a^\omega\} \cup \{a^n b a^\omega \mid n \geq 0\}$. However, $\{\lambda, a^\omega\} \;{}_{\{a\}}{\sqcup\!\sqcup}_{\{b\}}\; (\{\lambda, b^2\} \;{}_{\{b\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b\}) = \{\lambda, a^\omega\} \;{}_{\{a\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda\} = \{\lambda, a^\omega\}$. □

Hence we propose to add not just λ, but all prefixes of the words involved. In the following example we show that this solves the problems encountered in the previous examples.

Example 6.4.36. (Examples 6.4.33 and 6.4.35 continued) The problem we met in Example 6.4.33 is indeed solved in this way, viz. $(\{\lambda, a\} \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b\}) \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{c\}}\; (\{c^n \mid n \geq 0\} \cup \{c^\omega\}) = (\{\lambda, a\} \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{c\}}\; (\{c^n \mid n \geq 0\} \cup \{c^\omega\}) = \{c^n a c^\omega, c^m a c^n, c^n \mid m, n \geq 0\} \cup \{c^\omega\} = \{\lambda, a\} \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{b,c\}}\; (\{c^n b c^\omega, c^m b c^n, c^n \mid m, n \geq 0\} \cup \{c^\omega\}) = \{\lambda, a\} \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{b,c\}}\; (\{\lambda, b\} \;{}_{\{b\}}{\sqcup\!\sqcup}_{\{c\}}\; (\{c^n \mid n \geq 0\} \cup \{c^\omega\}))$.

Moreover, also the problem we met in Example 6.4.35 is indeed solved in this way, viz. $((\{a^n \mid n \geq 0\} \cup \{a^\omega\}) \;{}_{\{a\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b, b^2\}) \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b\} = (\{a^m b a^n b a^\omega, a^m b a^n b a^p, a^n b a^\omega, a^m b a^n, a^n \mid m, n, p \geq 0\} \cup \{a^\omega\}) \;{}_{\{a,b\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b\} = \{a^n b a^\omega, a^m b a^n, a^n \mid n \geq 0\} \cup \{a^\omega\} = (\{a^n \mid n \geq 0\} \cup \{a^\omega\}) \;{}_{\{a\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b\} = (\{a^n \mid n \geq 0\} \cup \{a^\omega\}) \;{}_{\{a\}}{\sqcup\!\sqcup}_{\{b\}}\; (\{\lambda, b, b^2\} \;{}_{\{b\}}{\sqcup\!\sqcup}_{\{b\}}\; \{\lambda, b\})$. □

This provides us with enough motivation to set out and prove associativity of (general, unfair) fS-shuffling and rS-shuffling at the level of prefix-closed languages. It is relevant to recall at this point that the behavior of a team automaton and that of its constituting component automata are prefix closed. Hence we can still apply this higher-level notion of associativity to behavior of team automata.

First we express (general) S-shuffles in terms of fair S-shuffles and prefixes (cf. Lemma 6.3.4).

Lemma 6.4.37. *Let Γ be an alphabet. Then*

(1) if $u \in \Delta_1^*$ and $v \in \Delta_2^*$, then $u \,\|^\Gamma\, v = u \,|||^\Gamma\, v$,

(2) if $u \in \Delta_1^*$ and $v \in \Delta_2^\omega$, then $u \,\|^\Gamma v = \bigcup_{u' \in pref(u),\, pres_\Gamma(u') = pres_\Gamma(u)} (u' \,|||^\Gamma v)$, and

(3) if $u \in \Delta_1^\omega$ and $v \in \Delta_2^\omega$, then $u \,\|^\Gamma v = \bigcup_{u' \in pref(u),\, pres_\Gamma(u') = pres_\Gamma(u)} (u' \,|||^\Gamma v) \cup \bigcup_{v' \in pref(v),\; pres_\Gamma(v') = pres_\Gamma(v)} (u \,|||^\Gamma\, v') \cup u \,|||^\Gamma\, v$.

Proof. (1) Trivial.

(2) Let $u \in \Delta_1^*$ and let $v \in \Delta_2^\omega$.

($\subseteq$) Let $w \in u \,||^{\Gamma}\, v$. By Definition 6.4.1, there exists an $n \geq 1$ such that $w \in (u_1 \,||\, v_1)x_1(u_2 \,||\, v_2)x_2 \cdots x_{n-1}(u_n \,||\, v_n)$, where $u_1, u_2, \ldots, u_n \in (\Delta_1 \setminus \Gamma)^*$, $v_1, v_2, \ldots, v_{n-1} \in (\Delta_2 \setminus \Gamma)^*$, $v_n \in (\Delta_2 \setminus \Gamma)^\omega$, $u = u_1x_1u_2x_2 \cdots x_{n-1}u_n$, and $v = v_1x_1v_2x_2 \cdots x_{n-1}v_n$. Then, according to Lemma 6.3.4(2), $u_n \,||\, v_n = \bigcup_{u' \in \mathrm{pref}(u_n)}(u' \,|||\, v)$ and hence we obtain $w \in (u_1 \,||\, v_1)x_1(u_2 \,||\, v_2)x_2 \cdots x_{n-1}(u_n \,||\, v_n) = \bigcup_{u' \in \mathrm{pref}(u_n)}((u_1 \,||\, v_1)x_1(u_2 \,||\, v_2)x_2 \cdots x_{n-1}(u' \,||\, v_n)) = \bigcup_{u' \in \mathrm{pref}(u_n)} (u_1x_1u_2x_2 \cdots u_{n-1}x_{n-1}u' \ |||^{\Gamma}\ v_1x_1v_2x_2 \cdots v_{n-1}x_{n-1}v_n) = \bigcup_{\bar{u} \in \mathrm{pref}(u),\ \mathrm{pres}_{\Gamma}(\bar{u}) = \mathrm{pres}_{\Gamma}(u)}(\bar{u} \,|||^{\Gamma}\, v)$.

($\supseteq$) This follows immediately from Definitions 6.3.1 and 6.4.1.

(3) Analogous to (2) but now using Lemma 6.3.4(3). $\square$

As a consequence we obtain a characterization of fS-shuffling in terms of prefixes and preserving homomorphisms.

Corollary 6.4.38. *(1) If $u \in \Delta_1^*$ and $v \in \Delta_2^*$, then $u \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ v = \{w \in (\Delta_1 \cup \Delta_2)^* \mid pres_{\Delta_1}(w) = u,\ pres_{\Delta_2}(w) = v\}$,*

(2) if $u \in \Delta_1^*$ and $v \in \Delta_2^\omega$, then $u \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \exists\, u' \in pref(u) :\ pres_{\Delta_2}(u') = pres_{\Delta_2}(u),\ pres_{\Delta_1}(w) = u',\ pres_{\Delta_2}(w) = v\}$, and

(3) if $u \in \Delta_1^\omega$ and $v \in \Delta_2^\omega$, then $u \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ v = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid pres_{\Delta_1}(w) = u,\ pres_{\Delta_2}(w) = v\} \cup \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \exists\, u' \in pref(u) :\ pres_{\Delta_2}(u') = pres_{\Delta_2}(u),\ pres_{\Delta_1}(w) = u',\ pres_{\Delta_2}(w) = v\} \cup \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \exists\, v' \in pref(v) :\ pres_{\Delta_1}(v') = pres_{\Delta_1}(v),\ pres_{\Delta_1}(w) = u,\ pres_{\Delta_2}(w) = v'\}$.

Proof. By Definition 6.4.3, $u \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ v = u \,||^{\Delta_1 \cap \Delta_2}\, v$ and $u \ {}_{\Delta_1}\underline{|||}_{\Delta_2}\ v = u \,|||^{\Delta_1 \cap \Delta_2}\, v$ whenever $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$. Moreover, for $x \in \Delta_1^\infty$, $\mathrm{pres}_{\Delta_1 \cap \Delta_2}(x) = \mathrm{pres}_{\Delta_2}(x)$ and, for $x \in \Delta_2^\infty$, $\mathrm{pres}_{\Delta_1 \cap \Delta_2}(x) = \mathrm{pres}_{\Delta_1}(x)$. The statements now follow by combining Corollary 6.4.27 and Lemma 6.4.37, with $\Gamma = \Delta_1 \cap \Delta_2$. $\square$

With this corollary we can now prove a result similar to Corollary 6.4.27, which was used to prove the associativity of fair fS-shuffling. In this case, however, we (have to) deal with words together with their prefixes.

Lemma 6.4.39. *If $u \in \Delta_1^\infty$ and $v \in \Delta_2^\infty$, then $(\{u\} \cup pref(u)) \ {}_{\Delta_1}\underline{||}_{\Delta_2}\ (\{v\} \cup pref(v)) = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid pres_{\Delta_1}(w) \in \{u\} \cup pref(u),\ pres_{\Delta_2}(w) \in \{v\} \cup pref(v)\}$.*

Proof. Let $u \in \Delta_1^\infty$ and let $v \in \Delta_2^\infty$. We distinguish three cases.

(1) If $u \in \Delta_1^*$ and $v \in \Delta_2^*$, then by Corollary 6.4.38(1), $(\{u\} \cup \mathrm{pref}(u)) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; (\{v\} \cup \mathrm{pref}(v)) = \{w \in (\Delta_1 \cup \Delta_2)^* \mid \mathrm{pres}_{\Delta_1}(w) \in \{u\} \cup \mathrm{pref}(u),\ \mathrm{pres}_{\Delta_2}(w) \in \{v\} \cup \mathrm{pref}(v)\}$.

(2) If $u \in \Delta_1^*$ and $v \in \Delta_2^\omega$, then the fact that $u \in \mathrm{pref}(u)$ implies that $(\{u\} \cup \mathrm{pref}(u)) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; (\{v\} \cup \mathrm{pref}(v)) = \mathrm{pref}(u) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; (\{v\} \cup \mathrm{pref}(v)) = (\mathrm{pref}(u) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; \mathrm{pref}(v)) \cup (\mathrm{pref}(u) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; \{v\})$. By Corollary 6.4.38(2), $\mathrm{pref}(u) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; \{v\} = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \exists\, u' \in \mathrm{pref}(u),\ u'' \in \mathrm{pref}(u') :\ \mathrm{pres}_{\Delta_2}(u') = \mathrm{pres}_{\Delta_2}(u''),\ \mathrm{pres}_{\Delta_1}(w) = u'',\ \mathrm{pres}_{\Delta_2}(w) = v\} = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \exists\, u'' \in \mathrm{pref}(u) :\ \mathrm{pres}_{\Delta_1}(w) = u'',\ \mathrm{pres}_{\Delta_2}(w) = v\}$. Combining this with (1), we obtain $\mathrm{pref}(u) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; (\{v\} \cup \mathrm{pref}(v)) = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \mathrm{pres}_{\Delta_1}(w) \in \mathrm{pref}(u),\ \mathrm{pres}_{\Delta_2}(w) \in \{v\} \cup \mathrm{pref}(v)\}$.

(3) If $u \in \Delta_1^\omega$ and $v \in \Delta_2^\omega$, then $(\{u\} \cup \mathrm{pref}(u)) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; (\{v\} \cup \mathrm{pref}(v)) = L_1 \cup L_2 \cup L_3$, with L_1, L_2, and L_3 as follows.
$L_1 = \mathrm{pref}(u) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; (\{v\} \cup \mathrm{pref}(v)) = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \mathrm{pres}_{\Delta_1}(w) \in \mathrm{pref}(u),\ \mathrm{pres}_{\Delta_2}(w) \in \{v\} \cup \mathrm{pref}(v)\}$ by (2).
$L_2 = (\{u\} \cup \mathrm{pref}(u)) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; \mathrm{pref}(v) = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \mathrm{pres}_{\Delta_1}(w) \in \{u\} \cup \mathrm{pref}(u),\ \mathrm{pres}_{\Delta_2}(w) \in \mathrm{pref}(v)\}$ by (2) and the commutativity of fS-shuffling.
$L_3 = u \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; v = \{w \in (\Delta_1 \cup \Delta_2)^\omega \mid \mathrm{pres}_{\Delta_1}(w) = u,\ \mathrm{pres}_{\Delta_2}(w) = v\} \cup L_1' \cup L_2'$, with $L_1' \subseteq L_1$ and $L_2' \subseteq L_2$, by Corollary 6.4.38(3).

Consequently, $(\{u\} \cup \mathrm{pref}(u)) \;{}_{\Delta_1}\underline{\|}_{\Delta_2}\; (\{v\} \cup \mathrm{pref}(v)) = \{w \in (\Delta_1 \cup \Delta_2)^\infty \mid \mathrm{pres}_{\Delta_1}(w) \in \{u\} \cup \mathrm{pref}(u),\ \mathrm{pres}_{\Delta_2}(w) \in \{v\} \cup \mathrm{pref}(v)\}$. □

We have thus found a characterization of fS-shuffling that is insensitive to the order of application.

Theorem 6.4.40. *Let $u_i \in \Delta_i^\infty$, for all $i \in [3]$. Then*

$$(\{u_1\} \cup \mathit{pref}(u_1))\;{}_{\Delta_1}\underline{\|}_{\Delta_2 \cup \Delta_3}\;((\{u_2\} \cup \mathit{pref}(u_2))\;{}_{\Delta_2}\underline{\|}_{\Delta_3}\;(\{u_3\} \cup \mathit{pref}(u_3))) =$$
$$\{w \in (\Delta_1 \cup \Delta_2 \cup \Delta_3)^\infty \mid \forall\, i \in [3] :\ \mathit{pres}_{\Delta_i}(w) \in \{u_i\} \cup \mathit{pref}(u_i)\}.$$

Proof. ($\subseteq$) Let $w \in (\{u_1\} \cup \mathrm{pref}(u_1)) \;{}_{\Delta_1}\underline{\|}_{\Delta_2 \cup \Delta_3}\; ((\{u_2\} \cup \mathrm{pref}(u_2)) \;{}_{\Delta_2}\underline{\|}_{\Delta_3}\; (\{u_3\} \cup \mathrm{pref}(u_3)))$. By Lemma 6.4.39, $\mathrm{pres}_{\Delta_1}(w) \in \{u_1\} \cup \mathrm{pref}(u_1)$ and there exists a $y \in (\{u_2\} \cup \mathrm{pref}(u_2)) \;{}_{\Delta_2}\underline{\|}_{\Delta_3}\; (\{u_3\} \cup \mathrm{pref}(u_3))$ such that $\mathrm{pres}_{\Delta_2 \cup \Delta_3}(w) = y$. Consequently, $\mathrm{pres}_{\Delta_2}(w) = \mathrm{pres}_{\Delta_2}(\mathrm{pres}_{\Delta_2 \cup \Delta_3}(w)) = \mathrm{pres}_{\Delta_2}(y)$, which by Lemma 6.4.39 is included in $\{u_2\} \cup \mathrm{pref}(u_2)$, and $\mathrm{pres}_{\Delta_3}(w) = \mathrm{pres}_{\Delta_3}(\mathrm{pres}_{\Delta_2 \cup \Delta_3}(w)) = \mathrm{pres}_{\Delta_3}(y)$, which by Lemma 6.4.39 is included in $\{u_3\} \cup \mathrm{pref}(u_3)$.

($\supseteq$) Let $w \in (\Delta_1 \cup \Delta_2 \cup \Delta_3)^\infty$ be such that $\mathrm{pres}_{\Delta_i}(w) \in \{u_i\} \cup \mathrm{pref}(u_i)$, for all $i \in [3]$. Now let $z = \mathrm{pres}_{\Delta_2 \cup \Delta_3}(w)$. Hence $\mathrm{pres}_{\Delta_2}(z) = \mathrm{pres}_{\Delta_2}(w)$ and $\mathrm{pres}_{\Delta_3}(z) = \mathrm{pres}_{\Delta_3}(w)$. By Corollary 6.4.27, $z \in \mathrm{pres}_{\Delta_2}(w) \;{}_{\Delta_2}\underline{\|\!|}_{\Delta_3}\; \mathrm{pres}_{\Delta_3}(w)$

and $w \in \mathrm{pres}_{\Delta_1}(w) \;{}_{\Delta_1}\underline{|||}_{\Delta_2 \cup \Delta_3}\; z \subseteq \mathrm{pres}_{\Delta_1}(w) \;{}_{\Delta_1}\underline{|||}_{\Delta_2 \cup \Delta_3}\; (\mathrm{pres}_{\Delta_2}(w) \;{}_{\Delta_2}\underline{|||}_{\Delta_3}\; \mathrm{pres}_{\Delta_3}(w)) \subseteq (\{u_1\} \cup \mathrm{pref}\,(u_1)) \;{}_{\Delta_1}\underline{|||}_{\Delta_2 \cup \Delta_3}\; ((\{u_2\} \cup \mathrm{pref}\,(u_2)) \;{}_{\Delta_2}\underline{|||}_{\Delta_3}\; (\{u_3\} \cup \mathrm{pref}\,(u_3))) \subseteq (\{u_1\} \cup \mathrm{pref}\,(u_1)) \;{}_{\Delta_1}\underline{||}_{\Delta_2 \cup \Delta_3}\; ((\{u_2\} \cup \mathrm{pref}\,(u_2)) \;{}_{\Delta_2}\underline{||}_{\Delta_3}\; (\{u_3\} \cup \mathrm{pref}\,(u_3)))$. $\square$

It is worth noticing that the proof of this theorem shows how unfair fS-shuffling can be translated into fair fS-shuffling by including prefixes. The associativity of fS-shuffling of prefix-closed languages now follows immediately.

Theorem 6.4.41. *Let $u \in \Delta_1^\infty$, $v \in \Delta_2^\infty$, and $w \in \Delta_3^\infty$. Then*

$$(\{u\} \cup \mathit{pref}(u)) \;{}_{\Delta_1}\underline{||}_{\Delta_2 \cup \Delta_3}\; ((\{v\} \cup \mathit{pref}(v)) \;{}_{\Delta_2}\underline{||}_{\Delta_3}\; (\{w\} \cup \mathit{pref}(w))) = ((\{u\} \cup \mathit{pref}(u)) \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; (\{v\} \cup \mathit{pref}(v))) \;{}_{\Delta_1 \cup \Delta_2}\underline{||}_{\Delta_3}\; (\{w\} \cup \mathit{pref}(w)).$$

Proof. This follows directly from Theorem 6.4.40 and the commutativity of fS-shuffling. $\square$

Theorem 6.4.42. *Let $L_i \subseteq \Delta_i^\infty$, for all $i \in [3]$, be prefix closed. Then*

$$L_1 \;{}_{\Delta_1}\underline{||}_{\Delta_2 \cup \Delta_3}\; (L_2 \;{}_{\Delta_2}\underline{||}_{\Delta_3}\; L_3) = (L_1 \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; L_2) \;{}_{\Delta_1 \cup \Delta_2}\underline{||}_{\Delta_3}\; L_3.$$

Proof. ($\subseteq$) Let $w \in L_1 \;{}_{\Delta_1}\underline{||}_{\Delta_2 \cup \Delta_3}\; (L_2 \;{}_{\Delta_2}\underline{||}_{\Delta_3}\; L_3)$. Then by definition there exist words $u_1 \in L_1$, $u_2 \in L_2$, and $u_3 \in L_3$ such that $w \in (\{u_1\} \cup \mathrm{pref}\,(u_1)) {}_{\Delta_1}\underline{||}_{\Delta_2 \cup \Delta_3} ((\{u_2\} \cup \mathrm{pref}\,(u_2)) {}_{\Delta_2}\underline{||}_{\Delta_3} (\{u_3\} \cup \mathrm{pref}\,(u_3)))$. Consequently, by Theorem 6.4.41, $w \in ((\{u_1\} \cup \mathrm{pref}\,(u_1)) \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; (\{u_2\} \cup \mathrm{pref}\,(u_2))) \;{}_{\Delta_1 \cup \Delta_2}\underline{||}_{\Delta_3}\; (\{u_3\} \cup \mathrm{pref}\,(u_3)) \subseteq (L_1 \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; L_2) \;{}_{\Delta_1 \cup \Delta_2}\underline{||}_{\Delta_3}\; L_3$ by the fact that L_1, L_2, and L_3 are prefix closed.

($\supseteq$) This follows from (1) and the commutativity of fS-shuffling. $\square$

As before in the case of fair rS-shuffling, the fact that the rS-shuffle is defined in terms of the S-shuffle, together with the associativity of fS-shuffling, allows us to conclude that also rS-shuffling of prefix-closed languages is associative.

Theorem 6.4.43. *Let Γ be an alphabet and let $u_i \in \Delta_i^\infty$, for all $i \in [3]$. Then*

$$(\{u_1\} \cup \mathit{pref}(u_1)) {}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2 \cup \Delta_3} ((\{u_2\} \cup \mathit{pref}(u_2)) {}_{\Delta_2}\underline{||}^{\Gamma}_{\Delta_3} (\{u_3\} \cup \mathit{pref}(u_3))) = ((\{u_1\} \cup \mathit{pref}(u_1)) \;{}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\; (\{u_2\} \cup \mathit{pref}(u_2))) \;{}_{\Delta_1 \cup \Delta_2}\underline{||}^{\Gamma}_{\Delta_3}\; (\{u_3\} \cup \mathit{pref}(u_3)).$$

Proof. Similar to the proof of Theorem 6.4.30 by renaming the symbols, but now using Theorem 6.4.41. $\square$

Theorem 6.4.44. *Let Γ be an alphabet and let $L_i \subseteq \Delta_i^\infty$, for all $i \in [3]$, be prefix closed. Then*

$$L_1 \;{}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2 \cup \Delta_3}\; (L_2 \;{}_{\Delta_2}\underline{||}^{\Gamma}_{\Delta_3}\; L_3) = (L_1 \;{}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\; L_2) \;{}_{\Delta_1 \cup \Delta_2}\underline{||}^{\Gamma}_{\Delta_3}\; L_3.$$

Proof. Analogous to the proof of Theorem 6.4.42. $\square$

6.4.4 Conclusion

The commutativity and associativity of the S-shuffle (cf. Theorems 6.4.17 and 6.4.21) directly imply that the order in which we (fair) S-shuffle — on an alphabet Γ — a number of languages, is irrelevant, i.e. $L_1 \mathbin{|||}^{\Gamma} L_2 \mathbin{|||}^{\Gamma} \cdots \mathbin{|||}^{\Gamma} L_n$ and $L_1 \mathbin{||}^{\Gamma} L_2 \mathbin{||}^{\Gamma} \cdots \mathbin{||}^{\Gamma} L_n$ unambiguously define the fair S-shuffle and the S-shuffle, respectively, on Γ of languages $L_1, L_2, \ldots, L_n$, for an $n \geq 1$. We introduce some shorthand notations for such *n-ary (fair) S-shuffles*.

Notation 13. *We denote the fair S-shuffle $L_1 \mathbin{|||}^{\Gamma} L_2 \mathbin{|||}^{\Gamma} \cdots \mathbin{|||}^{\Gamma} L_n$ and the S-shuffle $L_1 \mathbin{||}^{\Gamma} L_2 \mathbin{||}^{\Gamma} \cdots \mathbin{||}^{\Gamma} L_n$, for an $n \geq 1$, by $|||^{\Gamma}_{i\in[n]} L_i$ and $||^{\Gamma}_{i\in[n]} L_i$, respectively.* □

Note that contrary to the (fair) shuffle and the (fair) S-shuffle, it is currently impossible to write either the (fair) fS-shuffle or the (fair) rS-shuffle — on an alphabet Γ — of languages $L_1, L_2, \ldots, L_n$, for an $n \geq 3$, without brackets since the order in which they are applied determines the synchronization symbols. We now present an example to illustrate this.

Example 6.4.45. Let $L_1 \subseteq \Delta_1^*$, $L_2 \subseteq \Delta_2^*$, $L_3 \subseteq \Delta_3^*$, and $L_4 \subseteq \Delta_4^*$. Then by Theorem 6.4.29, $((L_1 \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; L_2) \;{}_{\Delta_1\cup\Delta_2}\underline{|||}_{\Delta_3}\; L_3) \;{}_{\Delta_1\cup\Delta_2\cup\Delta_3}\underline{|||}_{\Delta_4}\; L_4 = (L_1 \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; L_2) \;{}_{\Delta_1\cup\Delta_2}\underline{|||}_{\Delta_3\cup\Delta_4}\; (L_3 \;{}_{\Delta_3}\underline{|||}_{\Delta_4}\; L_4) = L_1 \;{}_{\Delta_1}\underline{|||}_{\Delta_2\cup\Delta_3\cup\Delta_4}\; (L_2 \;{}_{\Delta_2}\underline{|||}_{\Delta_3\cup\Delta_4}\; (L_3 \;{}_{\Delta_3}\underline{|||}_{\Delta_4}\; L_4))$.

Now we let Γ be an alphabet. Then $((L_1 \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; L_2) \;{}_{\Delta_1\cup\Delta_2}\underline{|||}^{\Gamma}_{\Delta_3}\; L_3) \;{}_{\Delta_1\cup\Delta_2\cup\Delta_3}\underline{|||}^{\Gamma}_{\Delta_4}\; L_4 = (L_1 \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; L_2) \;{}_{\Delta_1\cup\Delta_2}\underline{|||}^{\Gamma}_{\Delta_3\cup\Delta_4}\; (L_3 \;{}_{\Delta_3}\underline{|||}^{\Gamma}_{\Delta_4}\; L_4) = L_1 \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2\cup\Delta_3\cup\Delta_4}\; (L_2 \;{}_{\Delta_2}\underline{|||}^{\Gamma}_{\Delta_3\cup\Delta_4}\; (L_3 \;{}_{\Delta_3}\underline{|||}^{\Gamma}_{\Delta_4}\; L_4))$ by Theorem 6.4.31. □

There are various ways of writing the n-ary (fair) fS-shuffles and (fair) rS-shuffles, for an $n \geq 3$, which by Theorems 6.4.29, 6.4.31, 6.4.42, and 6.4.44, are equivalent — provided that in the unfair case the languages are prefix closed. We choose the left-associative variants as standard representants of these classes.

Notation 14. *Let $n \geq 1$.*

The fair fS-shuffle of languages $L_1, L_2, \ldots, L_n$, with respect to $\Delta_1, \Delta_2, \ldots, \Delta_n$, is $(\cdots(L_1 \;{}_{\Delta_1}\underline{|||}_{\Delta_2}\; L_2) \;{}_{\Delta_1\cup\Delta_2}\underline{|||}_{\Delta_3}\; \cdots) \;{}_{\bigcup_{i\in[n-1]}\Delta_i}\underline{|||}_{\Delta_n}\; L_n$ and the fS-shuffle of $L_1, L_2, \ldots, L_n$, with respect to $\Delta_1, \Delta_2, \ldots, \Delta_n$, is $(\cdots(L_1 \;{}_{\Delta_1}\underline{||}_{\Delta_2}\; L_2) \;{}_{\Delta_1\cup\Delta_2}\underline{||}_{\Delta_3}\; \cdots) \;{}_{\bigcup_{i\in[n-1]}\Delta_i}\underline{||}_{\Delta_n}\; L_n$.

The fair rS-shuffle on an alphabet Γ of $L_1, L_2, \ldots, L_n$, with respect to $\Delta_1, \Delta_2, \ldots, \Delta_n$, is $(\cdots(L_1 \;{}_{\Delta_1}\underline{|||}^{\Gamma}_{\Delta_2}\; L_2) \;{}_{\Delta_1\cup\Delta_2}\underline{|||}^{\Gamma}_{\Delta_3}\; \cdots) \;{}_{\bigcup_{i\in[n-1]}\Delta_i}\underline{|||}^{\Gamma}_{\Delta_n}\; L_n$ and the rS-shuffle on Γ of $L_1, L_2, \ldots, L_n$, with respect to $\Delta_1, \Delta_2, \ldots, \Delta_n$, is $(\cdots(L_1 \;{}_{\Delta_1}\underline{||}^{\Gamma}_{\Delta_2}\; L_2) \;{}_{\Delta_1\cup\Delta_2}\underline{||}^{\Gamma}_{\Delta_3}\; \cdots) \;{}_{\bigcup_{i\in[n-1]}\Delta_i}\underline{||}^{\Gamma}_{\Delta_n}\; L_n$. □

We now introduce some shorthand notations for these *n-ary (fair) fS-shuffles* and *n-ary (fair) rS-shuffles*.

Notation 15. *Let $n \geq 1$.*

We denote the fair fS-shuffle and the fS-shuffle of languages $L_1, L_2, \ldots, L_n$, with respect to $\Delta_1, \Delta_2, \ldots, \Delta_n$, by $\underline{|||}_{\{\Delta_i | i \in [n]\}} L_i$ and $\underline{||}_{\{\Delta_i | i \in [n]\}} L_i$, respectively.

We denote the fair rS-shuffle and the rS-shuffle on an alphabet Γ of $L_1, L_2, \ldots, L_n$, with respect to $\Delta_1, \Delta_2, \ldots, \Delta_n$, by $\underline{|||}^{\Gamma}_{\{\Delta_i | i \in [n]\}} L_i$ and $\underline{||}^{\Gamma}_{\{\Delta_i | i \in [n]\}} L_i$, respectively. □

For the next section it is convenient to reformulate some results on the associativity of (fair) fS-shuffling using the new notations.

Theorem 6.4.46. *(1) If $w_i \in \Delta_i^\infty$, for all $i \in [n]$, then $\underline{|||}_{\{\Delta_i | i \in [n]\}} \{w_i\} = \{w \in (\bigcup_{i \in [n]} \Delta_i)^\infty \mid \forall\, i \in [n] : \ pres_{\Delta_i}(w) = w_i\}$, and*

(2) if $L_i \subseteq \Delta_i^\infty$, for all $i \in [n]$, are prefix closed, then $\underline{||}_{\{\Delta_i | i \in [n]\}} L_i = \{w \in (\bigcup_{i \in [n]} \Delta_i)^\infty \mid \forall\, i \in [n] : \ pres_{\Delta_i}(w) \in L_i\}$.

Proof. (1) This follows from the repeated application of Corollary 6.4.27 and the observation that for all $i, j \in [n]$ and $x \in \Delta_i^\infty$, $\mathrm{pres}_{\Delta_i}(\mathrm{pres}_{\Delta_i \cup \Delta_j}(x)) = \mathrm{pres}_{\Delta_i}(x)$.

(2) This follows from Theorem 6.4.40 and its proof. □

6.5 Team Automata Satisfying Compositionality

In this section we combine the relations between the behavior of team automata and that of their constituting component automata — as developed in Section 6.2 — and the (synchronized) shuffles from Sections 6.3 and 6.4.

In our general setup team automata may have an infinite set of component automata. In the context of compositionality, however, it is more realistic to consider team automata composed over a finite set of component automata.

Notation 16. *For the remainder of this chapter we assume that our fixed composable system $\mathcal{S}$ is finite, viz. $\mathcal{I}$ is a finite subset of $\mathbb{N}$.* □

Each *ai* synchronization in a team automaton requires the participation of all its constituting component automata sharing the action being synchronized. This is reflected in the following result, which shows that the behavior of the *maximal-ai* team automaton, in which no *ai* synchronizations are excluded, can be described as the fS-shuffle of the behavior of its constituting component automata. Corresponding versions of this result have been formulated for other automata-based specification models with composition based on the *ai* principle (see, e.g., [Tut87] and [Jon87]).

Theorem 6.5.1. *Let $\mathcal{T}$ be the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$. Then*

$$\mathbf{B}_{\mathcal{T}}^{\Sigma,\infty} = \underline{||}_{\{\Sigma_i \mid i \in \mathcal{I}\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}.$$

Proof. ($\subseteq$) This follows immediately from Theorem 6.2.9, the prefix closure of the behavior of component automata, and Theorem 6.4.46(2).

($\supseteq$) Let $w \in \underline{||}_{\{\Sigma_i \mid i \in \mathcal{I}\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$. Note that each $\mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$ is prefix closed. Hence, according to Theorem 6.4.46(2), $\mathrm{pres}_{\Sigma_i}(w) \in \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$, for all $i \in \mathcal{I}$. Consequently, by definition there exist $\alpha_i \in \mathbf{C}_{\mathcal{C}_i}^{\infty}$ such that $\mathrm{pres}_{\Sigma_i}(\alpha_i) = \mathrm{pres}_{\Sigma_i}(w)$, for all $i \in \mathcal{I}$. Hence $\prod_{i\in\mathcal{I}} \alpha_i \in \prod_{i\in\mathcal{I}} \mathbf{C}_{\mathcal{C}_i}^{\infty}$. Since $w \in \Sigma^{\infty}$ is such that $\mathrm{pres}_{\Sigma_i}(w) = \mathrm{pres}_{\Sigma_i}(\alpha_i)$, for all $i \in \mathcal{I}$, Corollary 6.2.15 implies that there exists a $\beta \in \mathbf{C}_{\mathcal{T}}^{\infty}$ such that $\mathrm{pres}_{\Sigma}(\beta) = w$. Hence $w \in \mathbf{B}_{\mathcal{T}}^{\Sigma,\infty}$. □

Example 6.5.2. (Example 6.2.12 continued) Recall the $\mathcal{R}^{ai}$-team automaton $\mathcal{T}^{ai}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$, depicted in Figure 6.4(b).

Indeed we see that we get $\mathbf{B}_{\mathcal{T}^{ai}}^{\Sigma,\infty} = \{\lambda, a\} = (\{b^n \mid n \geq 0\} \cup \{b^n a \mid n \geq 0\} \cup \{b^\omega\}) \;_{\{a,b\}}\underline{||}_{\{a,b\}} (\{\lambda\} \cup \{ab^n \mid n \geq 0\} \cup \{ab^\omega\}) = \mathbf{B}_{\mathcal{C}_1}^{\Sigma_1,\infty} \;_{\Sigma_1}\underline{||}_{\Sigma_2} \mathbf{B}_{\mathcal{C}_2}^{\Sigma_2,\infty} = (\underline{||}_{\Sigma_1} \mathbf{B}_{\mathcal{C}_1}^{\Sigma_1,\infty}) \;_{\Sigma_1}\underline{||}_{\Sigma_2} \mathbf{B}_{\mathcal{C}_2}^{\Sigma_2,\infty} = \underline{||}_{\{\Sigma_i \mid i \in [2]\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$.

Now recall the team automaton $\mathcal{T}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$, depicted in Figure 6.3(a). Note that while $ba \notin \{\lambda, a\} = \underline{||}_{\{\Sigma_i \mid i \in [2]\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$, clearly $ba \in \mathbf{B}_{\mathcal{T}}^{\Sigma,\infty}$ □

Each *free* synchronization in a team automaton is such that only one of its component automata participates — under the assumption that a loop on the action being synchronized is always executed. Hence, if we require $\mathcal{S}$ to be loop limited, then the behavior of the *maximal-free* team automaton over $\mathcal{S}$ equals the shuffle of the behavior of the component automata from $\mathcal{S}$. Actually we prove a more general result, viz. that the behavior of a specific heterogeneous team automaton that is composed according to a mixture of *maximal-free* and *maximal-ai* synchronizations equals the rS-shuffle of the behavior of its constituting component automata.

Theorem 6.5.3. *Let $\bar{\Gamma} = \Sigma \setminus \Gamma$ and let $\mathcal{T}$ be the $\{\mathcal{R}_a^{ai} \mid a \in \Sigma \cap \Gamma\} \cup \{\mathcal{R}_a^{free} \mid a \in \bar{\Gamma}\}$-team automaton over $\mathcal{S}$. Then*

if $\mathcal{S}$ is $\bar{\Gamma}$-loop limited, then $\mathbf{B}_{\mathcal{T}}^{\Sigma,\infty} = \underline{||}^{\Gamma}_{\{\Sigma_i \mid i \in \mathcal{I}\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$.

Proof. Let $\mathcal{T}'$ be the team automaton that is obtained from $\mathcal{T}$ by attaching a label to each action from $\bar{\Gamma}$ depending on the component automaton executing that action, i.e. $\mathcal{T}' = (Q, \Sigma', \delta', I)$ with $\Sigma' = \{[a,i] \mid a \in \bar{\Gamma} \cap \Sigma_i,\ i \in \mathcal{I}\} \cup (\Sigma \cap \Gamma)$ and $\delta' = \{(q,[a,i],q') \mid a \in \bar{\Gamma},\ (q,a,q') \in \delta,\ \mathrm{proj}_i^{[2]}(q,q') \in \delta_{i,a},\ i \in \mathcal{I}\} \cup (\delta \cap (Q \times \Gamma \times Q))$. Since all actions from $\bar{\Gamma}$ are *free* in $\mathcal{T}$, the behavior of $\mathcal{T}$ is an encoding of the behavior of $\mathcal{T}'$. Let $\psi : (\Sigma')^* \to \Sigma^*$ be the homomorphism defined by $\psi([a,i]) = a$ and $\psi(a) = a$. Then clearly $\mathbf{B}_{\mathcal{T}}^{\Sigma,\infty} = \psi(\mathbf{B}_{\mathcal{T}'}^{\Sigma',\infty})$.

For all $i \in \mathcal{I}$, let $\mathcal{C}_i'$ be the component automaton that is obtained from $\mathcal{C}_i$ by labeling each of its actions from $\bar{\Gamma}$ with i, i.e. $\mathcal{C}_i' = (Q_i, \Sigma_i', \delta_i', I_i)$ with $\Sigma_i' = \{[a,i] \mid a \in \bar{\Gamma} \cap \Sigma_i\} \cup (\Gamma \cap \Sigma_i)$ and $\delta_i' = \{(q,[a,i],q') \mid a \in \bar{\Gamma},\ (q,a,q') \in \delta_i\} \cup (\delta_i \cap (Q_i \times \Gamma \times Q_i))$. Obviously, $\mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty} = \psi(\mathbf{B}_{\mathcal{C}_i'}^{\Sigma_i',\infty})$, for all $i \in \mathcal{I}$. Let $\mathcal{S}' = \{\mathcal{C}_i' \mid i \in \mathcal{I}\}$. Since $\mathcal{S}$ is $\bar{\Gamma}$-loop limited it thus follows that $(\delta')_{[a,i]} = \mathcal{R}_{[a,i]}^{\mathit{free}}(\mathcal{S}')$, for all $a \in \bar{\Gamma}$ and for all $i \in \mathcal{I}$. Hence $\mathcal{T}'$ is the $\{\mathcal{R}_a^{ai} \mid a \in (\Sigma \cap \Gamma)\} \cup \{\mathcal{R}_a^{\mathit{free}} \mid a \in \Sigma' \setminus \Gamma\}$-team automaton over $\mathcal{S}'$. Moreover, since the component automata from $\mathcal{S}'$ can share actions from $\Sigma \cap \Gamma$ but not from $\Sigma' \setminus \Gamma$, it follows that for all $K \subseteq \mathcal{I}$, $\bigcap_{k \in K} \Sigma_k' = \bigcap_{k \in K} \Sigma_k \cap \Gamma$. Hence Theorem 4.5.5 implies that $\mathcal{T}'$ is the *maximal-ai* team automaton over $\mathcal{S}'$ as well. Subsequently, Theorem 6.5.1 and Lemma 6.4.7(2) imply that $\mathbf{B}_{\mathcal{T}}^{\Sigma,\infty} = \psi(\mathbf{B}_{\mathcal{T}'}^{\Sigma',\infty}) = \psi(\underline{||}_{\{\Sigma_i' \mid i \in \mathcal{I}\}} \mathbf{B}_{\mathcal{C}_i'}^{\Sigma_i',\infty}) = \psi(\underline{||}^{\Gamma}_{\{\Sigma_i' \mid i \in \mathcal{I}\}} \mathbf{B}_{\mathcal{C}_i'}^{\Sigma_i',\infty})$, which equals $\underline{||}^{\Gamma}_{\{\psi(\Sigma_i') \mid i \in \mathcal{I}\}} \psi(\mathbf{B}_{\mathcal{C}_i'}^{\Sigma_i',\infty}) = \underline{||}^{\Gamma}_{\{\Sigma_i \mid i \in \mathcal{I}\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$ because $\psi(\Sigma' \setminus \Gamma) \cap \Gamma = \varnothing$. □

Example 6.5.4. (Example 6.2.1 continued) The $\mathcal{R}_a^{\mathit{free}} \cup \mathcal{R}_b^{ai}$-team automaton over $\{\mathcal{C}_1, \mathcal{C}_2\}$ is defined as $\mathcal{T}^{fa} = (\{(q_1,q_2),(q_1,q_2'),(q_1',q_2),(q_1',q_2')\},\{a,b\},\delta^{fa}, \{(q_1,q_2)\})$, where $\delta^{fa} = \{((q_1,q_2),a,(q_1,q_2')),((q_1,q_2),a,(q_1',q_2)),((q_1,q_2'),b,(q_1,q_2')),((q_1,q_2'),a,(q_1',q_2')),((q_1',q_2),a,(q_1',q_2'))\}$ and it is depicted in Figure 6.6(b).

Clearly $\{\mathcal{C}_1, \mathcal{C}_2\}$ is $\{a\}$-loop limited and indeed we see that $\mathbf{B}_{\mathcal{T}^{fa}}^{\Sigma,\infty} = \underline{||}^{\{b\}}_{\{\Sigma_i \mid i \in [2]\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$. □

The behavior of the *maximal-free* team automaton over a loop limited composable system thus equals the shuffle of the behavior of its constituting component automata.

Theorem 6.5.5. *Let $\mathcal{T}$ be the $\mathcal{R}^{\mathit{free}}$-team automaton over $\mathcal{S}$. Then*

if $\mathcal{S}$ is loop limited, then $\mathbf{B}_{\mathcal{T}}^{\Sigma,\infty} = ||_{i \in \mathcal{I}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$.

Proof. This follows immediately from Theorem 6.5.3 with $\Sigma \cap \Gamma = \varnothing$. □

Example 6.5.6. (Examples 6.2.22 and 6.5.4 continued) Recall the $\mathcal{R}^{\mathit{free}}$-team automaton $\mathcal{T}^{\mathit{free}}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$, depicted in Figure 6.6(a). Recall also that

$\{\mathcal{C}_1, \mathcal{C}_2\}$ is $\{a\}$-loop limited. Indeed $\mathbf{B}^{\{a\},\infty}_{\mathcal{T}^{free}} = \{\lambda, a, aa\} = \{\lambda, a\} \mid\mid \{\lambda, a\} = \mathbf{B}^{\{a\},\infty}_{\mathcal{C}_1} \mid\mid \mathbf{B}^{\{a\},\infty}_{\mathcal{C}_2} = (\mid\mid_{i\in[1]} \mathbf{B}^{\{a\},\infty}_{\mathcal{C}_1}) \mid\mid \mathbf{B}^{\{a\},\infty}_{\mathcal{C}_2} = \mid\mid_{i\in[2]} \mathbf{B}^{\{a\},\infty}_{\mathcal{C}_i}$.

Since $\{\mathcal{C}_1, \mathcal{C}_2\}$ however is not loop limited, it is no surprise that $ab \notin \mathbf{B}^{\Sigma,\infty}_{\mathcal{T}^{free}}$, whereas $ab \in \mid\mid_{i\in[2]} \mathbf{B}^{\Sigma_i,\infty}_{\mathcal{C}_i}$. □

Summarizing, we have thus been able to describe the behavior of three types of team automata in terms of the behavior of their constituting component automata (cf. Theorems 6.5.1, 6.5.3, and 6.5.5). However, we needed the condition of loop limitedness to avoid ambiguity with respect to the participation of component automata in case of loops. The reason is — once again — the maximal interpretation adopted in Section 4.2. In the next chapter we will show how to circumvent this problem by switching to vectors of actions.

The results of this section provide a semantic equivalent of the syntactic hierarchical results presented in Sections 4.3 and 5.2. Recall from those sections that every iterated team automaton over $\mathcal{S}$ can be considered as a team automaton directly composed over $\mathcal{S}$. Hence, if we construct only *maximal-ai* team automata, then the fact that fS-shuffling is associative for prefix-closed languages implies that the behavior of each such (iterated) *maximal-ai* team automaton equals the fS-shuffle of the behavior of its constituting component automata from $\mathcal{S}$. Such (iterated) *maximal-ai* team automata thus satisfy compositionality. A similar reasoning can be applied in case we consider (iterated) *maximal-free* team automata or (iterated) $\{\mathcal{R}^{ai}_a \mid a \in \Sigma \cap \Gamma\} \cup \{\mathcal{R}^{free}_a \mid a \in \Sigma \setminus \Gamma\}$-team automata over $\mathcal{S}$, where Γ is an alphabet. These satisfy compositonality in the sense that their behavior equals the shuffle or rS-shuffle, respectively, of the behavior of their constituting component automata from $\mathcal{S}$. We now illustrate this exposition by an example. Note that the fact that the distinction of input, output, and internal actions is irrelevant here allows us to deal with synchronized automata rather than team automata in this example.

Example 6.5.7. (Example 4.3.1 continued) Assume that all synchronized automata composed in Example 4.3.1 are *maximal-ai* synchronized automata.

Theorem 6.5.1 then implies that $\mathbf{B}^{\Sigma,\infty}_{\mathcal{T}_{1-7}} = \underline{||}_{\{\Sigma_i \mid i\in[7]\}} \mathbf{B}^{\Sigma_i,\infty}_{\mathcal{A}_i}$.

Consequently, together with the commutativity of fS-shuffling (cf. Corollary 6.4.19) and the associativity of fS-shuffling for prefix closed languages (cf. Theorems 6.4.29 and 6.4.42) Theorem 6.5.1 furthermore implies that $\mathbf{B}^{\Gamma'',\infty}_{\mathcal{T}''} = \mathbf{B}^{\Gamma',\infty}_{\mathcal{T}'} \ {}_{\cup_{i\in[6]}\Sigma_i}\underline{||}_{\Sigma_7} \ \mathbf{B}^{\Sigma_7,\infty}_{\mathcal{A}_7} = (\mathbf{B}^{\Gamma_1,\infty}_{\mathcal{T}_{\{2,4,6\}}} \ {}_{\cup_{i\in\{2,4,6\}}\Sigma_i}\underline{||}_{\cup_{i\in\{1,3,5\}}\Sigma_i} \ \mathbf{B}^{\Gamma_2,\infty}_{\mathcal{T}_{\{1,3,5\}}}) \ {}_{(\cup_{i\in\{2,4,6\}}\Sigma_i)\cup(\cup_{i\in\{1,3,5\}}\Sigma_i)}\underline{||}_{\Sigma_7} \ \mathbf{B}^{\Sigma_7,\infty}_{\mathcal{A}_7} = ((\underline{||}_{\{\Sigma_i \mid i\in\{2,4,6\}\}} \mathbf{B}^{\Sigma_i,\infty}_{\mathcal{A}_i}) \ {}_{\cup_{i\in\{2,4,6\}}\Sigma_i}\underline{||}_{\cup_{i\in\{1,3,5\}}\Sigma_i} \ (\underline{||}_{\{\Sigma_i \mid i\in\{1,3,5\}\}} \mathbf{B}^{\Sigma_i,\infty}_{\mathcal{A}_i})) \ {}_{(\cup_{i\in\{2,4,6\}}\Sigma_i)\cup(\cup_{i\in\{1,3,5\}}\Sigma_i)}\underline{||}_{\Sigma_7} \ \mathbf{B}^{\Sigma_7,\infty}_{\mathcal{A}_7} = (\underline{||}_{\{\Sigma_i \mid i\in[6]\}} \mathbf{B}^{\Sigma_i,\infty}_{\mathcal{A}_i} \ {}_{\cup_{i\in[6]}\Sigma_i}\underline{||}_{\Sigma_7} \ \mathbf{B}^{\Sigma_7,\infty}_{\mathcal{A}_7} = \underline{||}_{\{\Sigma_i \mid i\in[7]\}} \mathbf{B}^{\Sigma_i,\infty}_{\mathcal{A}_i} = \mathbf{B}^{\Sigma,\infty}_{\mathcal{T}_{1-7}}$. □

We close this chapter with an observation on *si* synchronizations. From a behavioral point of view, *si* synchronizations are very different from both *ai* and *free* synchronizations. While an *ai* synchronization of an action requires the participation of every component automaton with that action, a *free* synchronization of an action requires the participation of only and exactly one component automaton with that action. Whether an action of a component automaton is required to participate in an *si* synchronization of that action, however, cannot be decided without information on its current local state. A shuffle that would describe the behavior of a *maximal-si* team automaton in terms of the behavior of its constituting component automata should thus be a type of synchronized shuffle that — depending on local states of the component automata — is able to decide which actions of the component automata must be interleaved and which must be synchronized. This, however, seems impossible due to the simple fact that the behavior of component automata is stripped from all state information.

7. Team Automata, I/O Automata, Petri Nets

In the Introduction we have discussed team automata in the context of related models from the literature. In this chapter we provide a more detailed comparison of team automata with two specific models, viz. *Input/Output automata* (I/O automata for short) and (labeled) *Petri nets*.

In [Ell97] the model of I/O automata underlies the considerations which led to the introduction of team automata. Here we show how indeed I/O automata fit formally within the framework of team automata.

Next we study how team automata relate to *Individual Token Net Controllers* (ITNCs for short) — a particular model of vector labeled Petri nets from the theory of *Vector Controlled Concurrent Systems* (VCCSs for short) developed in [Kee96]. To this aim, we introduce the intermediate model of *vector team automata* which execute vectors of actions rather than ordinary actions. For a subclass of vector team automata we subsequently provide a translation into ITNCs which shows how team automata can be viewed as fitting in the VCCS framework.

Notation 17. *In this chapter we again assume a fixed, but arbitrary and possibly infinite index set $\mathcal{I} \subseteq \mathbb{N}$, which we will use to index the component automata involved. For each $i \in \mathcal{I}$, we again let $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$ be a fixed component automaton and we use Σ_i to denote its set of actions $\Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$. Moreover, we again let $\mathcal{S} = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$ be a fixed composable system and we let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a fixed team automaton over $\mathcal{S}$. We furthermore use Σ to denote its set of actions $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$, we use Σ_{ext} to denote its set of external actions $\Sigma_{inp} \cup \Sigma_{out}$, and we use Σ_{loc} to denote its set of locally-controlled actions $\Sigma_{out} \cup \Sigma_{int}$. Recall that $\mathcal{I} \subseteq \mathbb{N}$ implies that $\mathcal{I}$ is ordered by the usual $\leq$ relation on $\mathbb{N}$, thus inducing an ordering on $\mathcal{S}$, and that the $\mathcal{C}_i$ are not necessarily different. Finally, we again let Θ be an arbitrary but fixed alphabet disjoint from Q.* $\square$

7.1 I/O Automata

Team automata were defined to be an extension of I/O automata. In this section we show how I/O automata fit into the framework of team automata.

Originally I/O automata are defined in terms of automata, together with an associated equivalence relation over the set of actions used to define so-called *fair* computations. In [Tut87], I/O automata without such equivalence relations are called *safe I/O automata* and in [GSSL94] they are referred to as *unfair*. Here we are not concerned with fairness in the context of I/O automata and we only consider safe or unfair I/O automata, which we will simply refer to as I/O automata.

7.1.1 Definitions

An I/O automaton is an automaton together with a classification of its actions as input, output, and internal actions. Input and output actions form the interface between the automaton and its environment, including other I/O automata, whereas internal actions cannot be observed by the environment. With these considerations in mind, I/O automata are formally defined as component automata, but with the additional condition that they should be reactive, i.e. *input enabling*. This means that whatever the current state of the automaton, it is always capable of receiving any of its potential inputs.

Definition 7.1.1. *An* Input/Output automaton *(*I/O automaton *for short) is a component automaton* $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *such that*

$\mathcal{C}$ *is* Σ_{inp}*-enabling.* □

From a given set of I/O automata, a new I/O automaton may be constructed provided that this set satisfies two conditions. These conditions only relate to the role of the actions and do not use the property of input enabling.

First the I/O automata should form a composable system. Hence, as for the definition of a team automaton, it is required that the internal actions of any of the I/O automata belong uniquely to that I/O automaton.

Secondly, there is the idea that two I/O automata cannot be expected to synchronize on an action which is output for both of them. Rather than complicating the notion of composition itself, this is prohibited by the requirement that the output actions of the I/O automata should be disjoint. This means that every external action can be output in at most one of the I/O automata.

Definition 7.1.2. *The composable system* $\mathcal{S}$ *is a* compatible system *if for all* $i \in \mathcal{I}$,

$\Sigma_{i,out} \cap \bigcup_{j \in \mathcal{I} \setminus \{i\}} \Sigma_{j,out} = \varnothing$. □

Note that every subset of a compatible system is again a compatible system.

The composition of I/O automata into a new I/O automaton is defined by requiring those I/O automata sharing an action a to perform this a simultaneously (i.e. synchronize on a). The intention is that such a simultaneous execution models a communication from the I/O automaton of which a is an output action to the I/O automata of which a is an input action. In fact, the execution of an input action is thought of as the notification of the arrival of output either from within the new I/O automaton (i.e. from another I/O automaton) or from the environment. In terms of our framework this means that a team automaton is constructed in which every action is *ai*. Furthermore — although this is only implicit in the explanation — all synchronizations which do not violate this condition have to be included, which matches our maximality principle. Hence the constructed team automaton is unique.

Definition 7.1.3. *$\mathcal{T}$ is the* team I/O automaton *over $\mathcal{S}$ if*

(1) $\mathcal{S}$ is compatible,

(2) each $\mathcal{C}_i$, $i \in \mathcal{I}$, is an I/O automaton, and

(3) for all $a \in \Sigma_{ext}$,

$$\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}). \qquad \square$$

We thus note that, given a compatible system $\mathcal{S}$ of I/O automata, the unique team I/O automaton over $\mathcal{S}$ is the *maximal-ai* team automaton over $\mathcal{S}$.

Note that the condition that each of the component automata $\mathcal{C}_i$ in a compatible system $\mathcal{S}$ is input enabling (i.e. $\Sigma_{i,inp}$-enabling) guarantees that $\mathcal{S}$ is input enabling (i.e. Σ_{inp}-enabling). Theorem 4.6.8 then implies that for all $a \in \Sigma_{inp}$, every a-transition in every component automaton $\mathcal{C}_i$ is omnipresent in $\mathcal{T}$, after which Theorem 5.5.9 implies that the *maximal-ai* team automaton $\mathcal{T}$ over $\mathcal{S}$ is $\Sigma_{inp} \cap \Sigma_i$-enabling, for all $i \in \mathcal{I}$. Hence $\mathcal{T}$ is Σ_{inp}-enabling. The composition of the *maximal-ai* team automaton over a compatible system of input-enabling component automata thus preserves input enabling, from which it follows that every team I/O automaton is again an I/O automaton.

Theorem 7.1.4. *Every team I/O automaton is an I/O automaton.* □

It must be noted that the *maximal-ai* team automaton over a composable system which is not compatible may still be an I/O automaton, even though the team I/O automaton over such a composable system does not exist.

Surprisingly enough, the following example shows that *not* every subteam determined by some $J \subseteq \mathcal{I}$ of a team I/O automaton over $\mathcal{S}$ is an I/O automaton, let alone the team I/O automaton over $\{\mathcal{C}_j \mid j \in J\}$.

Example 7.1.5. The I/O automata $\mathcal{C}_1 = (\{p\}, (\{a\}, \varnothing, \varnothing), \{(p,a,p)\}, \{p\})$ and $\mathcal{C}_2 = (\{q\}, (\varnothing, \{a\}, \varnothing), \varnothing, \{q\})$ obviously form a compatible system. The team I/O automaton over $\{\mathcal{C}_1, \mathcal{C}_2\}$ is $\mathcal{T} = (\{(p,q)\}, (\varnothing, \{a\}, \varnothing), \varnothing, \{(p,q)\})$ and its subteam determined by $\{1\}$ is $SUB_{\{1\}} = (\{(p)\}, (\{a\}, \varnothing, \varnothing), \varnothing, \{(p)\})$, which clearly is not an I/O automaton. □

An auxiliary condition is thus needed to guarantee that the subteam determined by some $J \subseteq \mathcal{I}$ of a team I/O automaton $\mathcal{T}$ over $\mathcal{S}$ is the team I/O automaton over $\{\mathcal{C}_j \mid j \in J\}$. As we show next, it suffices to require that in all component automata from $\mathcal{S}$, all output actions that are actions for the subteam have at least one transition.[1]

Theorem 7.1.6. *Let $\mathcal{S}$ be a compatible system of I/O automata and let $\mathcal{T}$ be the team I/O automaton over $\mathcal{S}$. Let $J \subseteq \mathcal{I}$. Then*

> *if for all $a \in \Sigma_{out}$ and for all j such that $a \in \Sigma_{j,out}$, $\delta_{j,a} \neq \varnothing$, then $SUB_J(\mathcal{T})$ is the team I/O automaton over $\{\mathcal{C}_j \mid j \in J\}$.*

Proof. By Theorem 5.1.10, $SUB_J(\mathcal{T})$ is a team automaton over $\{\mathcal{C}_j \mid j \in J\}$, so we only have to prove that the three requirements of Definition 7.1.3 hold.

(1) This follows from the observation that every subset of a compatible system is again a compatible system.

(2) This follows from the fact that each of the component automata $\mathcal{C}_j$, with $j \in J$, is an I/O automaton.

(3) For all $a \in \Sigma_{out}$ and for all j such that $a \in \Sigma_{j,out}$, let $\delta_{j,a} \neq \varnothing$. Then we have to prove that for all $a \in \Sigma_{J,ext}$, $(\delta_J)_a = \mathcal{R}_a^{ai}(\{\mathcal{C}_j \mid j \in J\})$. Now let $a \in \Sigma_{J,ext}$. Since $\mathcal{T}$ is the team I/O automaton over $\mathcal{S}$, we know that $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$. We distinguish two cases.

If $a \in \Sigma_{inp}$, then the fact that $\mathcal{S}$ is Σ_{inp}-enabling, together with the fact that $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$, implies that $\delta_a \neq \varnothing$.

If $a \in \Sigma_{out}$, then the fact that for all j such that $a \in \Sigma_{j,out}$, we know that $\delta_{j,a} \neq \varnothing$, together with the fact that $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$, implies that $\delta_a \neq \varnothing$.
It now follows by Theorem 4.7.5(2) that $(\delta_J)_a = \mathcal{R}_a^{ai}(\{\mathcal{C}_j \mid j \in J\})$. □

[1] In [BEKR01a] and [BEKR03] we erroneously did not include this condition.

7.1.2 Iterated Composition

By Theorem 7.1.4, every team I/O automaton is an I/O automaton and hence can be used to iteratively define higher-level team I/O automata. This allows us to continue the considerations of Sections 4.3 and 5.2, but now for team I/O automata rather than synchronized automata and team automata, respectively. In particular we thus have to take into account that team I/O automata can only be formed over compatible systems of I/O automata.

Notation 18. *For the rest of this section we let $\mathcal{S}$ be a compatible system of I/O automata.* □

We begin by showing that compatibility is preserved when iteratively forming team I/O automata.

Lemma 7.1.7. *Let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, form a partition of $\mathcal{I}$. Let, for each $j \in \mathcal{J}$, $\mathcal{T}_j$ be the team I/O automaton over $\mathcal{S}_j = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. Then*

$\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is a compatible system.

Proof. Denote for each $\mathcal{T}_j$, $j \in \mathcal{J}$, by Γ_j its set of actions, by $\Gamma_{j,out}$ its output alphabet, and by $\Gamma_{j,int}$ its internal alphabet. By definition $\Gamma_{j,out} = \bigcup_{i \in \mathcal{I}_j} \Sigma_{i,out}$, $\Gamma_{j,int} = \bigcup_{i \in \mathcal{I}_j} \Sigma_{i,int}$, and $\Gamma_j = \bigcup_{i \in \mathcal{I}_j} \Sigma_i$, for all $j \in \mathcal{J}$. Then the compatibility of $\mathcal{S}$ implies that for all $i \in \mathcal{I}$, $\Sigma_{i,int} \cap \bigcup_{\ell \in \mathcal{I} \setminus \{i\}} \Sigma_\ell = \varnothing$ and $\Sigma_{i,out} \cap \bigcup_{\ell \in \mathcal{I} \setminus \{i\}} \Sigma_{\ell,out} = \varnothing$. Since the $\mathcal{I}_j$ are mutually disjoint it now follows immediately that for all $j \in \mathcal{J}$, $\Gamma_{j,int} \cap \bigcup_{\ell \in \mathcal{J} \setminus \{j\}} \Gamma_\ell = \varnothing$ and $\Gamma_{i,out} \cap \bigcup_{\ell \in \mathcal{J} \setminus \{j\}} \Gamma_{\ell,out} = \varnothing$. Hence $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is a compatible system. □

Given the compatible system $\mathcal{S}$, we can iteratively form team I/O automata until after a finite number of iterations all I/O automata in $\mathcal{S}$ have been used once. During the iteration, compatibility is preserved by the previous lemma, while Theorem 7.1.4 guarantees that all intermediate team I/O automata are I/O automata. Hence we can speak of an iterated team I/O automaton over $\mathcal{S}$ similar to the notion of an iterated team automaton over some composable system, as defined in Definition 5.2.2.

Definition 7.1.8. *$\mathcal{T}$ is an* iterated team I/O automaton over *$\mathcal{S}$ if either*

(1) $\mathcal{T}$ is the team I/O automaton over $\mathcal{S}$, or

(2) $\mathcal{T}$ is the team I/O automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where each $\mathcal{T}_j$ is an iterated team I/O automaton over the compatible system $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$, for some $\mathcal{I}_j \subseteq \mathcal{I}$, and $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$. □

The fact that team I/O automata are constructed according to the *maximal-ai* principle (cf. Definition 7.1.3(3)) guarantees that no transitions are lost during the iteration process. Therefore any iterated team I/O automaton over $\mathcal{S}$ coincides — upto a reordering — with the team I/O automaton over $\mathcal{S}$.

Theorem 7.1.9. *Let $\mathcal{T}$ be the team I/O automaton over $\mathcal{S}$ and let $\widehat{\mathcal{T}}$ be an iterated team I/O automaton over $\mathcal{S}$. Then*

$$\langle\langle \widehat{\mathcal{T}} \rangle\rangle_{\mathcal{S}} = \mathcal{T}.$$

Proof. If $\widehat{\mathcal{T}}$ is directly obtained from $\mathcal{S}$, i.e. without iteration, then $\langle\langle \widehat{\mathcal{T}} \rangle\rangle_{\mathcal{S}} = \widehat{\mathcal{T}}$. Since $\mathcal{T}$ is the unique team I/O automaton over $\mathcal{S}$, it then follows that $\langle\langle \widehat{\mathcal{T}} \rangle\rangle_{\mathcal{S}} = \widehat{\mathcal{T}} = \mathcal{T}$.

Now let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ be a partition of $\mathcal{I}$, for some $\mathcal{J} \subseteq \mathbb{N}$, and assume that $\widehat{\mathcal{T}}$ is the team I/O automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, with each $\mathcal{T}_j$ an iterated team I/O automaton over the compatible system $\mathcal{S}_j = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. With an inductive argument we assume that $\langle\langle \mathcal{T}_j \rangle\rangle_{\mathcal{S}_j}$ is the team I/O automaton over $\mathcal{S}_j$. Consequently, we can use Theorem 5.2.6(1) to conclude that there exists a team automaton $\mathcal{T}'$ over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ such that $\langle\langle \mathcal{T}' \rangle\rangle_{\mathcal{S}} = \mathcal{T}$. Hence $\mathcal{T}' = \widehat{\mathcal{T}}$ follows once we have shown that $\mathcal{T}'$ is the *maximal-ai* team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$.

We first observe that $\mathcal{T}'$ and $\mathcal{T}$ have — upto a reordering — the same transitions, since $\langle\langle \mathcal{T}' \rangle\rangle_{\mathcal{S}} = \mathcal{T}$.

Now assume that some external action a of $\mathcal{T}'$ is not *ai* in $\mathcal{T}'$ with respect to $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$. Then $\mathcal{T}'$ has an a-transition in which some $\mathcal{T}_j$ is not involved even though a belongs to the external alphabet of that $\mathcal{T}_j$. This action a then also belongs to the external alphabet of one of the component automata $\mathcal{C}_i \in \mathcal{S}_j$, while this component automaton is not involved in the given transition. Consequently, the external action a of $\mathcal{T}$ is not *ai* in $\mathcal{T}$ with respect to $\mathcal{S}_j = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$, a contradiction.

Next assume that $\mathcal{T}'$ misses, for some external action a, an a-transition $(q, q') \in \mathcal{R}_a^{ai}(\{\mathcal{T}_j \mid j \in \mathcal{J}\})$. By Theorem 7.1.6, for each $j \in \mathcal{J}$, $\langle\langle \mathcal{T}_j \rangle\rangle_{\mathcal{S}_j} = SUB_{\mathcal{I}_j}(\mathcal{T})$ is the *maximal-ai* team automaton over $\mathcal{S}_j = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. Since $\mathcal{T}$ is moreover the *maximal-ai* team automaton over $\mathcal{S} = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$, it follows that this a-transition (q, q') — after reordering — belongs to $\mathcal{R}_a^{ai}(\mathcal{S})$, the set of all a-transitions of $\mathcal{T}$, a contradiction with $\langle\langle \mathcal{T}' \rangle\rangle_{\mathcal{S}} = \mathcal{T}$.

Hence $\mathcal{T}'$ is the *maximal-ai* team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, which implies that $\mathcal{T}' = \widehat{\mathcal{T}}$. Consequently $\langle\langle \widehat{\mathcal{T}} \rangle\rangle_{\mathcal{S}} = \langle\langle \mathcal{T}' \rangle\rangle_{\mathcal{S}} = \mathcal{T}$, as required. □

From this theorem it follows that any team I/O automaton over a compatible system can be constructed iteratively in any order from the given component automata. The only difference between the directly constructed team I/O automaton and an iteratively constructed version may be in the ordering and grouping of the state space and the transition space. This implies in particular that — upto a reordering — there exists only one (iterated) team I/O automaton over any given compatible system of component automata.

7.1.3 Synchronizations

We have seen above that the unique team I/O automaton over a compatible system $\mathcal{S}$ of I/O automata is the *maximal-ai* team automaton over $\mathcal{S}$. This means that the knowledge on *maximal-ai* team automata we have acquired so far, can now be applied to team I/O automata.

Notation 19. *For the rest of this section we let $\mathcal{T}$ be the unique team I/O automaton over $\mathcal{S}$.* □

The fact that $\mathcal{T}$ is the *maximal-ai* team automaton over $\mathcal{S}$ implies that all of its actions are thus *ai* by Theorem 4.5.3(2). This provides a formal description of the idea we set out with in Subsection 7.1.1, viz. that output actions of an I/O automaton are always received by those I/O automata that have its input counterpart as an action. Since I/O automata are input-enabled, it is even the case that the I/O automaton in which an action a is output never has to wait until those I/O automata in which a is input are ready for the communication. It may however be the case that an external action appears only as an input action in $\mathcal{S}$. Then it is again an input action of $\mathcal{T}$ and can be used as such in a higher-level team I/O automaton.

By combining Theorem 5.3.15, Corollary 5.3.19, and Theorem 5.3.20 with the fact that all actions of $\mathcal{T}$ are *ai*, we moreover obtain that all input actions of $\mathcal{T}$ are *sipp* and *wipp*, while all its output actions are *sopp*, *wopp*, *ms*, *sms*, and *wms*. This provides a formal description of the fact that indeed — as mentioned in the Introduction — peer-to-peer and master-slave types of synchronization cannot be distinguished in (team) I/O automata: all of their output actions are by definition *sopp* and *sms* (and thus also *wopp*, *wms*, and *ms*).

7.1.4 Behavior

In Chapter 6 we have presented some results on the behavior of finite component automata. One of the conclusions was that the distinction of the set of actions into input, output, and internal actions has no influence on the behavior of finite component automata. Here we investigate whether that situation changes as a result of the extra requirement that input actions need to be input enabling.

We denote $\mathsf{IOCA} = \{\mathbf{B}^{\Gamma}_{\mathcal{C}} \mid \Gamma$ is an alphabet and $\mathcal{C}$ is a finite input-enabling component automaton with full alphabet $\Gamma\}$. Then Lemma 6.1.1 and Definition 7.1.1 immediately yield $\mathsf{IOCA} \subseteq \mathsf{CA} = \mathsf{pREG}$. As was the case for the inclusion $\mathsf{pREG} \subseteq \mathsf{CA}$, the inclusion $\mathsf{pREG} \subseteq \mathsf{IOCA}$ can be proven by choosing any distribution of the alphabet of an automaton over input, output, and internal alphabets, but now input enabling should be guaranteed. Thus automaton $\mathcal{A} = (P, \Gamma, \gamma, J)$ such that P and Γ are finite can be viewed as a component automaton $\mathcal{C} = (P, (\Gamma_1, \Gamma_2, \Gamma_3), \gamma, J)$ such that $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 = \Gamma$, with $\mathbf{B}^{\Gamma}_{\mathcal{C}} = \mathbf{B}^{\Gamma}_{\mathcal{A}}$. Obviously, $\mathcal{C}$ is input enabling whenever $\Gamma_1 = \varnothing$. Hence we have the following result.

Lemma 7.1.10. $\mathsf{pREG} = \mathsf{CA} = \mathsf{IOCA}$. $\square$

From Lemma 6.1.2 we now conclude that all languages in IOCA can also be obtained as input, output, internal, external, and locally-controlled behavior of component automata.

We denote $\mathsf{IOCA}^{alph} = \{\mathbf{B}^{alph}_{\mathcal{C}} \mid \mathcal{C}$ is a finite input-enabling component automaton$\}$, with $alph \in \{inp, out, int, ext, loc\}$. Since all languages in IOCA^{alph} are the images under a weak coding of languages in $\mathsf{IOCA} = \mathsf{pREG}$, using the closure of pREG under weak codings we immediately obtain the following extension of Lemma 6.1.3.

Lemma 7.1.11. *Let* $alph \in \{inp, out, int, ext, loc\}$. *Then*

$$\mathsf{IOCA}^{alph} \subseteq \mathsf{pREG}. \qquad \square$$

Recall the component automata $[\mathcal{C}, out] = (P, (\varnothing, \Gamma, \varnothing), \gamma, J)$ and $[\mathcal{C}, int] = (P, (\varnothing, \varnothing, \Gamma), \gamma, J)$ as variants of an arbitrary component automaton $\mathcal{C} = (P, (\Gamma_{inp}, \Gamma_{out}, \Gamma_{int}), \gamma, J)$ with $\Gamma = \Gamma_{inp} \cup \Gamma_{out} \cup \Gamma_{int}$. Then we obtain the following result by combining Lemma 7.1.11 with Lemmata 7.1.10 and 6.1.2 and the observation that $[\mathcal{C}, out]$ and $[\mathcal{C}, int]$ trivially are input enabling.

Theorem 7.1.12. $\mathsf{pREG} = \mathsf{CA} = \mathsf{CA}^{inp} = \mathsf{CA}^{out} = \mathsf{CA}^{int} = \mathsf{CA}^{ext} = \mathsf{CA}^{loc} = \mathsf{IOCA} = \mathsf{IOCA}^{out} = \mathsf{IOCA}^{int} = \mathsf{IOCA}^{ext} = \mathsf{IOCA}^{loc}$. $\square$

Thus what remains to be done is to compare IOCA^{inp} and IOCA (or, equivalently, IOCA^{inp} and pREG). From Lemma 7.1.11 we already know that $\mathsf{IOCA}^{inp} \subseteq \mathsf{pREG}$. That this inclusion is proper follows immediately from the following lemma.

Lemma 7.1.13. *Let* $\mathcal{C} = (P, (\Gamma_{inp}, \Gamma_{out}, \Gamma_{int}), \gamma, J)$ *be a finite input-enabling component automaton such that* $J \neq \varnothing$. *Then*

$$\mathbf{B}^{inp}_{\mathcal{C}} = \Gamma^*_{inp}.$$

Proof. $\mathbf{B}_{\mathcal{C}}^{inp} \subseteq \Gamma_{inp}^{*}$ is trivial, while $\Gamma_{inp}^{*} \subseteq \mathbf{B}_{\mathcal{C}}^{inp}$ follows directly from the fact that $\mathcal{C}$ is input enabling, i.e. for all states $p \in P$ and for all input actions $a \in \Gamma_{inp}$ there exists a transition $(p, a, p') \in \gamma$. □

Theorem 7.1.14. $\mathsf{IOCA}^{inp} \subset \mathsf{pREG}$. □

Finally, since $\mathcal{T}$ is the *maximal-ai* team automaton over $\mathcal{S}$, Theorem 6.5.1 immediately implies that its behavior equals the fS-shuffle of the behavior of its constituting I/O automata provided that $\mathcal{S}$ is finite.

Theorem 7.1.15. *If $\mathcal{I}$ is finite, then* $\mathbf{B}_{\mathcal{T}}^{\Sigma,\infty} = \underline{\|}_{\{\Sigma_i \mid i \in \mathcal{I}\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i,\infty}$. □

A corresponding version of this result has been formulated for I/O automata (see, e.g., [Tut87]).

7.1.5 Conclusion

We have seen that from a structural viewpoint the I/O automaton model fits seamlessly in the framework of team automata. Results and notions from team automata thus become available for I/O automata. In particular, a framework is provided in which the underlying concepts of I/O automata can be given a broader perspective and compared with other ideas. For instance, the possibility to define the language of a team I/O automaton directly — without actually considering the team — from the languages of its components (cf. Theorem 7.1.15) is an important property in the theory of I/O automata. The results presented in Subsection 6.5 now show that — while sufficient — composition based on *maximal-ai* synchronizations is not necessary to guarantee this property. Furthermore, the idea of subteams and iterative construction only marginally investigated for I/O automata, is now immediately available from the team automata framework.

Team automata allow more types of synchronization than I/O automata, however, which is very convenient when formally designing a system. In fact, for some designs it may be a disadvantage that the composition of I/O automata implies that output actions can always be traced back to a unique sender. In [Tut87], Tuttle writes "For instance [...] suppose that we construct automata modeling humans, and an automaton modeling a vending machine. Humans can insert coins into the vending machine (output from humans and input to the vending machine). Since we require that the output actions of automata in a composition be disjoint, if we compose a collection of humans with the vending machine, each human's output action of inserting a coin must be tagged with an identifier. Thus, the vending machine is effectively able to determine which human is inserting a coin, which is not necessarily a realistic model of this simple interaction. It might be interesting to study other notions of composition that would avoid this problem."

We conclude this section by demonstrating how the problem sketched by Tuttle can be avoided by using team automata rather than I/O automata. The more general framework of team automata allows us to profit from the freedom to consider a composable system that is not compatible (i.e. component automata may share output actions) and to *choose* the transition relation when constructing a team automaton.

Example 7.1.16. (Example 5.1.7 continued) Let $\mathcal{A}' = (\{s', t'\}, (\{c\}, \{\$\}, \varnothing), \{(s', \$, t'), (t', c, s')\}, \{s'\})$ be a component automaton modeling yet another coffee **a**ddict. It is essentially a copy of our coffee addict $\mathcal{A}$ depicted in Figure 5.2. Note that $\{\mathcal{C}, \mathcal{A}, \mathcal{A}'\}$ is a composable system.

We now show how our two coffee addicts can both obtain coffee from our vending machine by forming a team automaton $\mathcal{T}'$ over $\{\mathcal{C}, \mathcal{A}, \mathcal{A}'\}$. As before we only have to choose the transition relation of $\mathcal{T}'$. Our vending machine is very simple and handles one customer at a time. When one of our addicts throws in a dollar our vending machine gives him or her a coffee. This can be repeated ad infinitum.

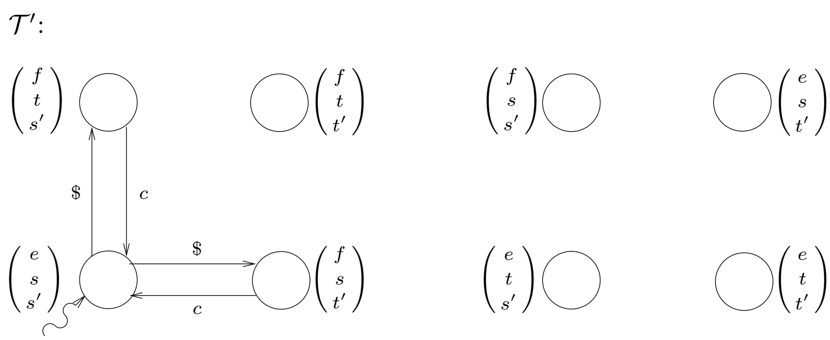


Fig. 7.1. Team automaton $\mathcal{T}'$ over $\{\mathcal{C}, \mathcal{A}, \mathcal{A}'\}$.

Formally, $\mathcal{T}'$ is defined as $\mathcal{T}' = (Q', (\varnothing, \{\$, c\}, \varnothing), \delta', \{(e, s, s')\})$, where $Q' = \{(e, s, s'), (e, s, t'), (e, t, s'), (e, t, t'), (f, s, s'), (f, s, t'), (f, t, s'), (f, t, t')\}$ and $\delta' = \{((e, s, s'), \$, (f, t, s')), ((f, t, s'), c, (e, s, s')), ((e, s, s'), \$, (f, s, t')), ((f, s, t'), c, (e, s, s'))\}$. It is depicted in Figure 7.1.

We see that the fact that the output actions of component automata need not be disjoint when constructing a team automaton allows us to equip both coffee addicts with the same output action \$ of inserting a coin. The freedom to choose a team automaton's transition relation moreover allows us to still model the action of one of the coffee addicts inserting a coin as different from the action of the other coffee addict inserting a coin. This contrasts with composition as defined for I/O automata, in which case the action \$ of inserting a coin would be modeled as a synchronized action of both coffee addicts. This surely would be highly unrealistic, as none of the coffee vending machines we have ever taken our coffee from allows the simultaneous insertion of two coins!

Note, finally, that the behavior of the vending machine in $\mathcal{T}'$ contains $\$c$, from which cannot be deduced whether it was coffee addict $\mathcal{A}$ or rather coffee addict $\mathcal{A}'$ that inserted a coin and was served a coffee. As noted by Tuttle himself, in case of I/O automata the vending machine would display either $\$_1c$ or $\$_2c$, in which case the identifiers thus indicate which coffee addict was served a coffee. □

7.2 Petri Nets

We now turn to a comparison of team automata with (labeled) *Petri nets*.

Petri nets were introduced in [Pet62] as a framework for modeling distributed systems. Since then they have been studied extensively (for an overview see, e.g., [RR98a] and [RR98b]). They occupy a central place in the study of distributed systems and are often used as a yardstick for other models.

Actually, "Petri net" is a generic name for a whole class of net-based models, consisting of an underlying structure (a net) together with rules describing its dynamics. Within a net one distinguishes places (which represent local aspects of the global states) and transitions (representing actions). To avoid confusion with the transitions of automata, we will from here on refer to Petri net transitions as events. Events are connected to places and places to events. Thus a net is a bipartite directed graph. In some models, certain elements may be labeled. The dynamics of a net is given in the form of rules defining when (in which states) an event can occur and its effect on the current state if it occurs. It is fundamental to Petri net theory that both the conditions allowing an event to occur and the effect of an occurrence on the global state, are local in the sense that they only involve places in the immediate neighborhood of (adjacent to) the event.

Team automata model distributed systems composed of component automata which work together by synchronizing their transitions. A synchronization describes the effect of a global (team) action on a global state of the system in terms of the local state changes of the component automata simultaneously executing that action. Component automata not involved in a synchronization remain idle and their current state is not affected.

Team automata thus resemble a (labeled) Petri net model with the local states as places and the synchronizations as labeled events. In a team automaton, executing an action (as the occurrence of a labeled transition) has a local effect, restricted to the local states of those component automata that are actually involved in executing that action. However, the synchronizations in a team automaton are selected from the complete transition spaces of their actions, which implies that the enabling of an action in the team automaton depends on the current global state as a whole rather than just on the local states of the component automata that are about to execute that action. This concept is called *state sharing* in [EG02], in which team automata are compared with UML statecharts (see, e.g. [UML99]) as used in object-oriented modeling. Moreover, due to the loop problem it is not always clear which component automata exactly take part in a synchronization.

In this section we first turn to the latter problem and propose a switch from (team) actions to vectors of (component) actions from which the participation of a component automaton in a synchronization can be seen immediately. This switch then makes it possible to view (vector) team automata as *Vector Controlled Concurrent Systems* (VCCSs for short) and, in particular, to relate a subclass of (vector) team automata to *Individual Token Net Controllers* (ITNCs for short) — a model of vector labeled Petri nets developed within the VCCS framework.

7.2.1 Vector Actions and Vector Team Automata

By the definition of team automata, each transition of $\mathcal{T}$ is of the form (q, a, q') with $a \in \Sigma$ and $q, q' \in \prod_{i \in \mathcal{I}} Q_i$. We now switch from transitions (q, a, q') to *vector transitions* $(q, \underline{a}, q')$, where $\underline{a}$ is an element of $\prod_{i \in \mathcal{I}} \{a, \lambda\}$, i.e. $\underline{a}$ is a *vector action*, with for each component automaton a corresponding entry which is either a or λ. If an entry of $\underline{a}$ is a, then this indicates that the corresponding component automaton takes part in the synchronization on a, while if it is λ, then that component automaton is not involved.

This switch to vector transitions and vector actions is feasible since for each transition (q, a, q') the global state change from q to q', caused by the occurrence of this transition, is described in terms of changes in the local states of the component automata involved. Let $j \in \mathcal{I}$. If $\mathrm{proj}_j(q) \neq \mathrm{proj}_j(q')$, then $\mathrm{proj}_j^{[2]}(q, q') \in \delta_{j,a}$ and the j-th component automaton is involved. In that case we set $\mathrm{proj}_j(\underline{a}) = a$. If $\mathrm{proj}_j(q) = \mathrm{proj}_j(q')$ and $\mathrm{proj}_j^{[2]}(q, q') \notin \delta_{j,a}$, then the j-th component automaton is not involved and we set $\mathrm{proj}_j(\underline{a}) = \lambda$. There is however — again — the problem of loops. If $\mathrm{proj}_j(q) = \mathrm{proj}_j(q')$

and $\mathrm{proj}_j^{[2]}(q,q') \in \delta_{j,a}$, then it is unclear whether or not the j-th component automaton is involved. Following the maximal interpretation as adopted in Section 4.2 we assume it is and thus set $\mathrm{proj}_j(\underline{a}) = a$.

Following the above procedure we can now transform an "ordinary" team automaton (over $\mathcal{S}$) into a *vector team automaton* (over $\mathcal{S}$), which thus has vector transitions and vector actions rather than "flat" transitions and "flat" actions.

On the other hand, one may also directly define a vector team automaton over $\mathcal{S}$ by describing the required synchronizations straight away as transitions with vectors as labels. In that case, for each action a, one chooses vector transitions from the *complete vector transition space* $\Delta_a^v(\mathcal{S})$ of a in $\mathcal{S}$ describing all possible vector transitions for a.

Definition 7.2.1. *Let $a \in \Sigma$. Then the* complete vector transition space *of a in $\mathcal{S}$ is denoted by $\Delta_a^v(\mathcal{S})$ and is defined as*

$$\begin{aligned}\Delta_a^v(\mathcal{S}) = \{(q,\underline{a},q') \mid {} & (q,q') \in \Delta_a(\mathcal{S}) \wedge \underline{a} \in \textstyle\prod_{i\in\mathcal{I}}\{a,\lambda\} \wedge (\exists i \in \mathcal{I} : \\ & proj_i(\underline{a}) \neq \lambda) \wedge (\forall i \in \mathcal{I} : [proj_i(\underline{a}) = a] \Rightarrow [proj_i^{[2]}(q,q') \in \delta_{i,a}] \wedge \\ & [proj_i(\underline{a}) = \lambda] \Rightarrow [proj_i(q) = proj_i(q')])\}. \quad \square\end{aligned}$$

If $(q,\underline{a},q') \in \Delta_a^v(\mathcal{S})$, then $\underline{a}$ is called a *vector representation* of a in $\mathcal{S}$ or a *vector action* of $\mathcal{S}$. Observe that due to the fact that $\mathcal{S}$ is composable, every internal action has at most one vector representation and this representation has exactly one entry which is not λ. Furthermore, all vector representations of external actions have *at least* one entry which is not λ.

A vector team automaton over $\mathcal{S}$ is now defined exactly as an ordinary team automaton over $\mathcal{S}$, except that its transition relation consists of vector transitions defining state changes caused by vector actions.

Definition 7.2.2. *A* vector team automaton over $\mathcal{S}$ *is a construct $\mathcal{T}^v = (Q,(\Sigma_{inp},\Sigma_{out},\Sigma_{int}),\delta^v,I)$, where*

$$\delta^v \subseteq \textstyle\bigcup_{a\in\Sigma} \Delta_a^v(\mathcal{S}) \text{ and moreover } \Delta_a^v(\mathcal{S}) \subseteq \delta^v \text{ if } a \in \Sigma_{int}. \qquad \square$$

We call δ^v the set of *(labeled) vector transitions* of $\mathcal{T}^v$ and we define $\delta_{\underline{a}}^v = \{(q,q') \mid (q,\underline{a},q') \in \delta^v\}$ as the set of *vector $\underline{a}$-transitions* of $\mathcal{T}^v$.

Completely analogous to the way we extracted subteams from team automata, we can distinguish a *subteam* within $\mathcal{T}^v$ by focusing on a subset of $\mathcal{S}$. Its vector actions and vector transitions are restrictions of the vector actions and vector transitions of $\mathcal{T}^v$ to the component automata in the subteam.

Definition 7.2.3. *Let $\mathcal{T}^v = (Q,(\Sigma_{inp},\Sigma_{out},\Sigma_{int}),\delta^v,I)$ be a vector team automaton over $\mathcal{S}$ and let $J \subseteq \mathcal{I}$. Then the* subteam of $\mathcal{T}^v$ determined by J

is denoted by $SUB_J(\mathcal{T}^v)$ *and is defined as* $SUB_J(\mathcal{T}^v) = (Q_J, (\Sigma_{J,inp}, \Sigma_{J,out}, \Sigma_{J,int}), \delta^v_J, I_J)$*, where*

$Q_J = \prod_{j\in J} Q_j$,
$\Sigma_{J,inp} = (\bigcup_{j\in J} \Sigma_{j,inp}) \setminus \bigcup_{j\in J} \Sigma_{j,out}$,
$\Sigma_{J,out} = \bigcup_{j\in J} \Sigma_{j,out}$,
$\Sigma_{J,int} = \bigcup_{j\in J} \Sigma_{j,int}$,
$\delta^v_J = \{ (proj_J(q), proj_J(\underline{a}), proj_J(q')) \in \Delta^v_a(\{\mathcal{C}_j \mid j \in J\}) \mid (q, \underline{a}, q') \in \delta^v \}$*, and*
$I_J = \prod_{j\in J} I_j$. □

It is not hard to see that a subteam of a vector team automaton satisfies the requirements of a vector team automaton.

Theorem 7.2.4. *Let* $\mathcal{T}^v$ *be a vector team automaton over* $\mathcal{S}$ *and let* $J \subseteq \mathcal{I}$*. Then*

$SUB_J(\mathcal{T}^v)$ *is a vector team automaton over* $\{\mathcal{C}_j \mid j \in J\}$*.*

Proof. Analogous to the proof of Theorem 5.1.10. □

The definition of (finite and infinite) computations of vector team automata can be carried over from (team) automata in the obvious way. This we will do later in this section, when we will also propose definitions for the behavior of vector team automata. We first concentrate on the structure of the synchronizations of vector team automata.

Before illustrating some of the notions introduced above, we make the following two remarks. First note that whenever the distinction of the alphabet into input, output, and internal actions is irrelevant, then a synchronized automaton can be seen as a team automaton. As a matter of fact, in examples in this chapter we will often refer to synchronized automata defined in earlier chapters as team automata. Secondly, recall that vectors may be written vertically as well as horizontally. In figures we will more often write them vertically, even though in the text they are more often written horizontally.

Example 7.2.5. (Example 4.2.1 continued) Consider vector team automata $\mathcal{T}^v_1 = (Q, (\varnothing, \{a\}, \varnothing), \delta^v_1, \{(p, q, r)\})$ and $\mathcal{T}^v_2 = (Q, (\varnothing, \{a\}, \varnothing), \delta^v_2, \{(p, q, r)\})$, in which $Q = \{ (p, q, r), (p, q, r') \}$ and $\delta^v_1 = \{ ((p, q, r), (\lambda, \lambda, a), (p, q, r')), ((p, q, r'), (\lambda, a, \lambda), (p, q, r')) \}$, whereas $\delta^v_2 = \{ ((p, q, r), (\lambda, a, a), (p, q, r')), ((p, q, r'), (\lambda, a, \lambda), (p, q, r'))\}$, over the composable system $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ obtained by turning the automata $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ into component automata with output actions only. $\mathcal{T}^v_1$ and $\mathcal{T}^v_2$ are depicted in Figure 7.2.

Note that in both vector team automata for each vector transition it is clear which component automata participate. This contrasts with the team automaton $\mathcal{T}$ depicted in Figure 4.6(b).

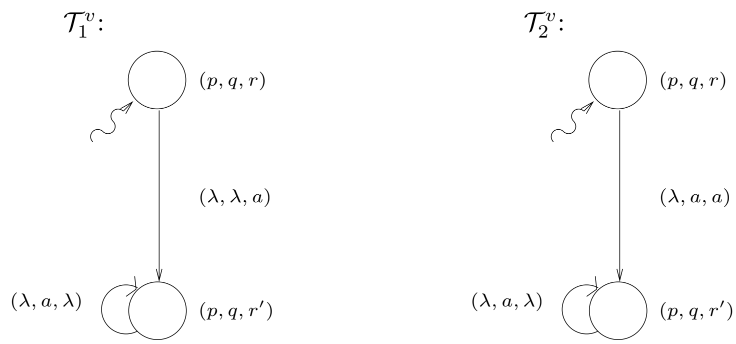


Fig. 7.2. Vector team automata $\mathcal{T}_1^v$ and $\mathcal{T}_2^v$.

Finally, it is clear that the subteam $SUB_{\{1\}}(\mathcal{T}_1^v) = SUB_{\{1\}}(\mathcal{T}_2^v) = (\{(p)\}, (\varnothing, \varnothing, \varnothing), \varnothing, \{(p)\})$ has the same structure as $\mathcal{A}_1$, depicted in Figure 4.6(a), whereas the subteam $SUB_{\{2,3\}}(\mathcal{T}_1^v) = (\{(q,r), (q,r')\}, (\varnothing, \{a\}, \varnothing), \{((q,r), (\lambda, a), (q,r')), ((q,r'), (a,\lambda), (q,r'))\}, \{(q,r)\})$ is depicted in Figure 7.3. □

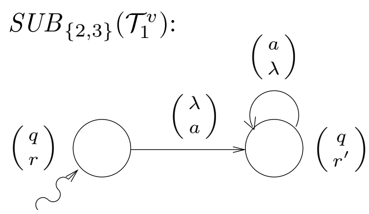


Fig. 7.3. Subteam $SUB_{\{2,3\}}(\mathcal{T}_1^v)$ of vector team automaton $\mathcal{T}_1^v$.

By replacing each transition $(q, \underline{a}, q')$ of a vector team automaton $\mathcal{T}^v$ by the flat transition (q, a, q') whenever $\underline{a}$ is a vector representation of the action a, one obtains the *flattened version* of $\mathcal{T}^v$.

Definition 7.2.6. *Let* $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ *be a vector team automaton. Then the* flattened version of $\mathcal{T}^v$ *is denoted by* $\mathcal{T}_F^v$ *and is defined as* $\mathcal{T}_F^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta_F^v, I)$*, where*

$$\delta_F^v = \{(q, a, q') \mid \exists\, \underline{a} \in \textstyle\prod_{i \in \mathcal{I}} \{a, \lambda\} : (q, \underline{a}, q') \in \delta^v\}. \qquad \square$$

The flattened version of a vector team automaton is an ordinary team automaton with essentially the same synchronizations as the vector team automaton.

Lemma 7.2.7. *The flattened version of a vector team automaton over $\mathcal{S}$ is a team automaton (over $\mathcal{S}$).*

Proof. Follows directly from Definitions 5.1.6, 7.2.2, and 7.2.6. □

Note that whereas each vector team automaton has a single flattened version, many vector team automata may define the same flattened version.

Example 7.2.8. (Example 7.2.5 continued) Both vector team automaton $\mathcal{T}_1^v$ and vector team automaton $\mathcal{T}_2^v$ have team automaton $\mathcal{T}$, as depicted in Figure 4.6(b), as their flattened version. □

Due to flattening, the explicit information on the execution of loops is lost. In this sense vector team automata thus have more modeling power than ordinary team automata.

We now present a more elaborate example that illustrates the advantage of vector actions as regards modeling explicit information on loops.

Example 7.2.9. (Example 5.3.2 continued) We show how to form a vector team automaton from W_1 and W_2. Let $\mathcal{T}^v_{\{1,2\}}$ be the vector team automaton $\mathcal{T}^v_{\{1,2\}} = (\{(s_1,s_2),(s_1,t_2),(t_1,s_2),(t_1,t_2)\},(\varnothing,\{a,b\},\varnothing),\delta^v_{\{1,2\}},\{(s_1,s_2)\})$, where $\delta^v_{\{1,2\}} = \{((s_1,s_2),(b,b),(s_1,s_2)),((s_1,s_2),(a,a),(t_1,t_2)),((t_1,t_2),(a,a),(t_1,t_2)),((t_1,t_2),(b,b),(s_1,s_2))\}$, over $\{W_1,W_2\}$. It is depicted in Figure 7.4.

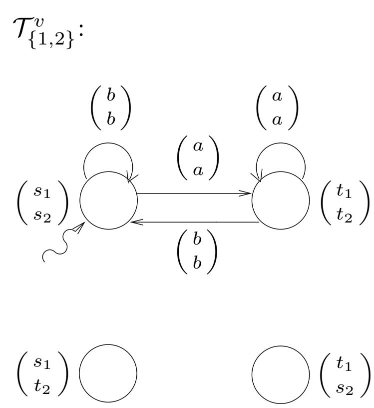


Fig. 7.4. Vector team automaton $\mathcal{T}^v_{\{1,2\}}$.

Clearly its flattened version $(\mathcal{T}^v_{\{1,2\}})_F$ equals the team automaton $\mathcal{T}_{\{1,2\}}$, depicted in Figure 4.1(a). Note however that $\mathcal{T}^v_{\{1,2\}}$ contains explicit information on loops that $(\mathcal{T}^v_{\{1,2\}})_F = \mathcal{T}_{\{1,2\}}$ lacks. Consider, e.g., vector a-transition $((t_1,t_2),(a,a),(t_1,t_2)) \in \mathcal{T}^v_{\{1,2\}}$. One immediately sees that both

(t_1, a, t_1) and (t_2, a, t_2) were executed. From the corresponding a-transition $((t_1, t_2), a, (t_1, t_2)) \in (\mathcal{T}^v_{\{1,2\}})_F$, however, one can only conclude that at least one of the a-transitions (t_1, a, t_1) and (t_2, a, t_2) was executed and under the maximal interpretation we do assume that both wheels participate in this acceleration.

If we assume that a flat tire is modeled by disabled acceleration, then the vector transition $((t_1, t_2), (a, \lambda), (t_1, t_2))$ models the fact that wheel W_2 does not participate in this acceleration, i.e. the axle contains a flat tire. This information is lost by flattening as this transforms the vector transition $((t_1, t_2), (a, \lambda), (t_1, t_2))$ into $((t_1, t_2), a, (t_1, t_2))$, which is also the result of flattening $((t_1, t_2), (a, a), (t_1, t_2))$ in $\mathcal{T}^v_{\{1,2\}}$. □

7.2.2 Effect of Vector Synchronizations

The lack of explicit information on loops in ordinary team automata led us to adopt a maximal interpretation of the involvement of component automata in team transitions. In fact, the definitions of *free*, *ai*, and *si* actions in Section 4.4 are based on this maximal interpretation.

Recall that intuitively an action a is a *free* action of $\mathcal{T}$ if in each a-transition of $\mathcal{T}$ only one component automaton participates. Hence — as a consequence of the maximal interpretation — in Definition 4.4.1 the set of *free* actions of $\mathcal{T}$ is defined as $Free(\mathcal{T}) = \{a \in \Sigma \mid (q, q') \in \delta_a \Rightarrow \#\{i \in \mathcal{I} \mid a \in \Sigma_i \wedge \mathrm{proj}_i^{[2]}(q, q') \in \delta_{i,a}\} = 1\}$, i.e. a is a *free* action of $\mathcal{T}$ if in each a-transition of $\mathcal{T}$ only one component automaton *is able to* participate in this execution of a.

Example 7.2.10. For $i \in \{1, 2\}$, let $\mathcal{C}_i = (\{q_i\}, (\varnothing, \{a\}, \varnothing), \{(q_i, a, q_i)\}, \{q_i\})$ be two component automata. They are depicted in Figure 7.5(a).

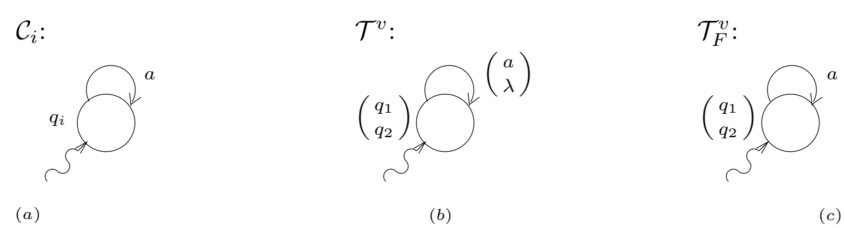


Fig. 7.5. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, vector team automaton $\mathcal{T}^v$, and its flattened version $\mathcal{T}^v_F$.

Clearly $\{\mathcal{C}_1, \mathcal{C}_2\}$ is a composable system. Consider the vector team automaton $\mathcal{T}^v = (\{(q_1, q_2)\}, (\varnothing, \{a\}, \varnothing), \{((q_1, q_2), (a, \lambda), (q_1, q_2))\}, \{(q_1, q_2)\})$

over $\{\mathcal{C}_1, \mathcal{C}_2\}$, depicted in Figure 7.5(b), and its flattened version $\mathcal{T}_F^v = (\{(q_1, q_2)\}, (\varnothing, \{a\}, \varnothing), \{((q_1, q_2), a, (q_1, q_2))\}, \{(q_1, q_2)\})$, which is depicted in Figure 7.5(c).

According to the definition, a is not *free* in $\mathcal{T}_F^v$. This is due to the fact that both component automata are able to participate in the a-transition of $\mathcal{T}_F^v$. In $\mathcal{T}^v$, however, action a is a *free* action in the sense that only component automaton $\mathcal{C}_1$ participates in the execution of a. □

Also the definitions of *ai* and *si* actions are based on the maximal interpretation. Intuitively, an action a is an *ai* (*si*) action of $\mathcal{T}$ if in each a-transition of $\mathcal{T}$ every component automaton participates which has a in its alphabet (provided that a is currently enabled in that component automaton). Formally, the set of *ai* actions of $\mathcal{T}$ is defined as $AI(\mathcal{T}) = \{a \in \Sigma \mid \forall i \in \mathcal{I} : (a \in \Sigma_i \wedge (q, q') \in \delta_a) \Rightarrow \mathrm{proj}_i^{[2]}(q, q') \in \delta_{i,a}\}$ and the set of *si* actions of $\mathcal{T}$ is defined as $SI(\mathcal{T}) = \{a \in \Sigma \mid \forall i \in \mathcal{I} : (a \in \Sigma_i \wedge (q, q') \in \delta_a \wedge a \ \mathrm{en}_{\mathcal{C}_i} \ \mathrm{proj}_i(q)) \Rightarrow \mathrm{proj}_i^{[2]}(q, q') \in \delta_{i,a}\}$. Hence, these definitions use the assumption that loops are executed, i.e. the maximal interpretation. Note that as a consequence, to determine whether or not an action a is *ai* (*si*) it suffices to consider for each a-transition of the team automaton only those component automata that do not have an a-loop at their current local state and check if they participate.

Example 7.2.11. (Example 7.2.10 continued) Action a is *ai* (and thus *si*) in $\mathcal{T}_F^v$ since $(q_i, q_i) \in \delta_{i,a}$, for all $i \in [2]$. On the contrary, in $\mathcal{T}^v$ we would *not* see a as an *si* action and (thus) neither as an *ai* action, since in $\mathcal{C}_2$ a is enabled at q_2 but $\mathcal{C}_2$ is not involved in $((q_1, q_2), (a, \lambda), (q_1, q_2))$. □

We now define when we consider a vector action to be *free*, *ai*, and *si* in a vector team automaton.

Definition 7.2.12. *Let $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ be a vector team automaton over $\mathcal{S}$. Then*

(1) the set of truly free *actions of $\mathcal{T}^v$ is denoted by tFree($\mathcal{T}^v$) and is defined as*

$$tFree(\mathcal{T}^v) = \{a \in \Sigma \mid (q, q') \in \delta^v_{\underline{a}} \Rightarrow \#\{i \in \mathcal{I} \mid proj_i(\underline{a}) = a\} = 1\},$$

(2) the set of truly ai *actions of $\mathcal{T}^v$ is denoted by $tAI(\mathcal{T}^v)$ and is defined as*

$$tAI(\mathcal{T}^v) = \{a \in \Sigma \mid \forall i \in \mathcal{I} : (a \in \Sigma_i \wedge (q, q') \in \delta^v_{\underline{a}}) \Rightarrow proj_i(\underline{a}) = a\}, \text{ and}$$

(3) the set of truly si *actions of $\mathcal{T}^v$ is denoted by $tSI(\mathcal{T}^v)$ and is defined as*

$$tSI(\mathcal{T}^v) = \{a \in \Sigma \mid \forall i \in \mathcal{I} : (a \in \Sigma_i \wedge (q, q') \in \delta^v_{\underline{a}} \wedge a \ en_{\mathcal{C}_i} \ proj_i(q)) \Rightarrow proj_i(\underline{a}) = a\}. \ \square$$

From this definition it follows immediately that every action that is truly *ai* in a vector team automaton is also truly *si*. Moreover, internal actions are always both truly *free* and truly *ai*. This reflects the properties of *free*, *ai*, and *si* in team automata as formulated in Lemmata 4.4.7 and 5.3.12.

Lemma 7.2.13. *Let* $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ *be a vector team automaton. Then*

(1) $tAI(\mathcal{T}^v) \subseteq tSI(\mathcal{T}^v)$ *and*

(2) $\Sigma_{int} \subseteq tFree(\mathcal{T}^v) \cap tAI(\mathcal{T}^v)$. □

Actions which are *free* in the flattened version of a vector team automaton are — due to the maximal interpretation — always truly *free* in that vector team automaton. The converse in general does not hold, as can be concluded from Example 7.2.10. For (truly) *ai* and *si* actions the situation is reversed. Actions which are truly *ai* (truly *si*) in a vector team automaton are always *ai* (*si*) in its flattened version. The converse in not true in general, as can be concluded from Example 7.2.11. The reason resides in the fact that the vector team automaton is not necessarily defined on basis of the maximal interpretation used for its flattened version.

Lemma 7.2.14. *Let* $\mathcal{T}^v$ *be a vector team automaton. Then*

(1) $Free(\mathcal{T}^v_F) \subseteq tFree(\mathcal{T}^v)$,

(2) $tAI(\mathcal{T}^v) \subseteq AI(\mathcal{T}^v_F)$, *and*

(3) $tSI(\mathcal{T}^v) \subseteq SI(\mathcal{T}^v_F)$. □

We have thus demonstrated with *free*, *ai*, and *si* as examples, how to interpret notions related to synchronizations in team automata for vector team automata. This would be a first step when developing a theory for synchronizations in vector team automata. In the rest of this section we continue our investigation of the relation between team automata and Petri nets, using vector team automata as the more general representatives within which the participation of component automata in synchronizations is made explicit.

7.2.3 Vector Controlled Concurrent Systems

The framework of VCCSs — originally introduced in [KKR90] and [KKR91] and further developed in [Kee96] — can be used to model concurrent systems consisting of a finite number of sequential components working together by synchronizing their actions. The basic idea is to use vectors both to specify the elementary synchronizations within a system and to describe its behavior. The approach has been inspired by the vector firing sequence semantics of path expressions and COSY (see, e.g., [Shi79] and [JL92]). It is related to the work of Arnold and Nivat (see, e.g. [Arn82]) and to the coordination of cooperating automata by synchronization on multisets as studied in [BDQT99].

A VCCS is specified by a description of (the sequential computations or behavior of) its constituting components and a control mechanism. The latter determines which combinations of sequential computations are allowed as the (concurrent) system's computations. To this aim, synchronization vectors specify which combinations of — possibly different — actions from the components may occur together. In addition, the control mechanism prescribes which and when synchronizations are available during the evolution of a system's computation.

It is immediate from the definitions that the synchronizations which can take place during a computation of a (vector) team automaton depend on the current state of the system. Hence (vector) team automata appear to fit in the VCCS framework. Before going into the details, we fix first some terminology and notation regarding vectors.

Let $J \subseteq \mathbb{N}$ be a finite and nonempty set of integers. Let $n = \#J$ be the cardinality of J. Let, for each $j \in J$, Δ_j be an alphabet. A vector $v \in \prod_{j \in J} \Delta_j^*$ is called an (*n-dimensional*) *word vector* (over $\{\Delta_j \mid j \in J\}$). We let $\Lambda = (\lambda, \ldots, \lambda) \in \prod_{j \in J} \Delta_j^*$ be the (*n-dimensional*) *empty word vector*, its dimension being clear from the context. A set of word vectors (over $\{\Delta_j \mid j \in J\}$) is called an (*n-dimensional*) *vector language* (over $\{\Delta_j \mid j \in J\}$).

A vector $w \in \prod_{j \in J}(\Delta_j \cup \{\lambda\}) \setminus \{\Lambda\}$ is called an (*n-dimensional*) *vector letter* (over $\{\Delta_j \mid j \in J\}$). The set of all vector letters over $\{\Delta_j \mid j \in J\}$ is denoted by $\mathrm{tot}\,(\{\Delta_j \mid j \in J\})$ and it is called the *total vector alphabet* over $\{\Delta_j \mid j \in J\}$. An *n-dimensional vector alphabet* (over $\{\Delta_j \mid j \in J\}$) is a subset of $\mathrm{tot}\,(\{\Delta_j \mid j \in J\})$. For a vector alphabet $\Delta \subseteq \mathrm{tot}\,(\{\Delta_j \mid j \in J\})$ we let Δ^u denote its subset of *uniform* vector letters over $\{\Delta_j \mid j \in J\}$, i.e. $\Delta^u = \{\underline{a} \in \Delta \mid \underline{a} \in \prod_{j \in J}\{a, \lambda\},\ a \in \bigcup_{j \in J} \Delta_j\}$. Since vector alphabets are alphabets, all terminology and notation for alphabets, words, and languages is carried over.

The *component-wise concatenation* of two n-dimensional vector letters $v = \prod_{j \in J} v_j$ and $w = \prod_{j \in J} w_j$ is defined by $v \circ w = \prod_{j \in J} v_j w_j$. For a vector alphabet $\Delta \subseteq \mathrm{tot}\,(\{\Delta_j \mid j \in J\})$ we define the homomorphism $\mathrm{coll}_{\Delta} : \Delta^* \to \prod_{j \in J} \Delta_j^*$ by $\mathrm{coll}_{\Delta}(v_1 v_2 \cdots v_k) = v_1 \circ v_2 \circ \cdots \circ v_k$, with $k \geq 0$. Thus, e.g.,

$\mathrm{coll}\left(\begin{pmatrix}\lambda\\a\end{pmatrix}\begin{pmatrix}b\\c\end{pmatrix}\begin{pmatrix}d\\\lambda\end{pmatrix}\right) = \begin{pmatrix}\lambda\\a\end{pmatrix}\circ\begin{pmatrix}b\\c\end{pmatrix}\circ\begin{pmatrix}d\\\lambda\end{pmatrix} = \begin{pmatrix}bd\\ac\end{pmatrix}$. This is the *collapse* of a sequence of vector letters from Δ into a word vector. The subscript Δ is omitted if it is clear from the context.

We now have the necessary terminology available to define the (finite and infinite) computations and the (finitary and infinitary) (vector) behavior of vector team automata. However, our vector letters and vector languages are of finite dimension.

Notation 20. *For the rest of this chapter we assume that $\mathcal{S}$ is a finite and nonempty composable system, i.e. $\mathcal{I}$ is a finite subset of* $\mathbb{N}$. □

Consequently we define $\mathrm{und}(\mathcal{T}^v) = (Q, \mathrm{tot}\,(\{\Sigma_i \mid i \in \mathcal{I}\}), \delta^v, I)$ to be the *underlying vector automaton* of $\mathcal{T}^v$.

For a given vector team automaton $\mathcal{T}^v$, its set of (*finite* and *infinite*) *computations* and its (*finitary* and *infinitary*) *behavior* are now defined as carried over from Definitions 3.1.2 and 3.1.7 through its underlying vector automaton $\mathrm{und}(\mathcal{T}^v)$. This means that we have, e.g., $\mathbf{C}_{\mathcal{T}^v} = \mathbf{C}_{\mathrm{und}(\mathcal{T}^v)}$ and $\mathbf{B}^{\Sigma}_{\mathcal{T}^v} = \mathbf{B}^{\Sigma}_{\mathrm{und}(\mathcal{T}^v)} = \mathrm{pres}_{\mathrm{tot}\,(\{\Sigma_i \mid i\in\mathcal{I}\})}(\mathbf{C}_{\mathcal{T}^v})$.

Due to the fact that vector team automata have vectors as actions it is possible to define also the (*finitary* and *infinitary*) *vector behavior* of a vector team automaton $\mathcal{T}^v$ as the collapse of the sequences of vector letters forming its (finitary and infinitary) behavior.

Definition 7.2.15. *Let $\mathcal{T}^v$ be a vector team automaton. Then*

(1) the finitary vector behavior *of $\mathcal{T}^v$ is denoted by $\mathbf{V}_{\mathcal{T}^v}$ and is defined as $\mathbf{V}_{\mathcal{T}^v} = \mathit{coll}(\mathbf{B}^{\Sigma}_{\mathcal{T}^v})$,*

(2) the infinitary vector behavior *of $\mathcal{T}^v$ is denoted by $\mathbf{V}^{\omega}_{\mathcal{T}^v}$ and is defined as $\mathbf{V}^{\omega}_{\mathcal{T}^v} = \mathit{coll}(\mathbf{B}^{\Sigma,\omega}_{\mathcal{T}^v})$, and*

(3) the vector behavior *of $\mathcal{T}^v$ is denoted by $\mathbf{V}^{\infty}_{\mathcal{T}^v}$ and is defined as $\mathbf{V}^{\infty}_{\mathcal{T}^v} = \mathit{coll}(\mathbf{B}^{\Sigma,\infty}_{\mathcal{T}^v}) = \mathbf{V}_{\mathcal{T}^v} \cup \mathbf{V}^{\omega}_{\mathcal{T}^v}$.* □

We now conclude that by Theorem 3.1.6 the finite computations of a vector team automaton determine its set of infinite computations and, by Theorem 3.1.5 and Corollary 3.1.11, also its (finitary and infinitary) behavior. Consequently, statements involving infinite computations and (finitary and infinitary) behavior of vector team automata can be proven by considering finite computations only.

7.2.4 Individual Token Net Controllers

Within the framework of VCCS, ITNCs have been defined as a particular type of control mechanism with an operational motivation. They are (finite) Petri nets, designed to follow and control the progress of the components of a system using individual tokens, one for each component. These tokens are distributed over the places, thus indicating the local state of each of the components. The global states of the net are then vectors of places, with each entry corresponding to a component. These distributions of the individual tokens over the places will be called markings here. The events of an ITNC model synchronizations between components. To be able to occur, an event needs certain individual tokens as input from its adjacent places and when it occurs it produces the same tokens as output in (in general) other places. In this way, the individual tokens used by an event determine which components take part in the synchronization. The events are labeled with vector letters with an entry for each component. Such an entry is empty if and only if the corresponding component does not take part in the synchronization (label-consistency). If it is not empty, then the corresponding component participates by executing the action mentioned. Note that these vector letters are not necessarily uniform. As will become clear, an ITNC can be interpreted as being built from a finite number of finite automata, each determined by one of the individual tokens. It is precisely this property that we will use in our translation of a subclass of (finite) vector team automata into ITNCs. We begin by formalizing the intuitive description given above.

Notation 21. *For the rest of this chapter we let* $n \geq 1$. □

An (n-dimensional) Individual Token Net Controller (n-ITNC or ITNC for short) consists of an underlying *(n-dimensional) Vector Labeled Individual Token Net* (n-VLITN or VLITN for short) together with a complete set of *initial markings* and a complete set of *final markings*. Such an n-VLITN is a labeled net with a specified set of n individual tokens.

Definition 7.2.16. *An* n-VLITN *is a construct* $\mathcal{N} = (P, T, O, F, V, \ell)$, *where*

- P *is the finite set of* places *of* $\mathcal{N}$,
- T *is the finite set of* events *of* $\mathcal{N}$ *such that* $P \cap T = \varnothing$,
- $O \subseteq \mathbb{N}$ *is a finite and nonempty set of* n *integers, called the set of* tokens *of* $\mathcal{N}$,
- $F : (P \times T) \cup (T \times P) \rightarrow \{o \mid o \subseteq O\}$ *is the* flow function *of* $\mathcal{N}$ *assigning subsets of* O *to elements of* $(P \times T) \cup (T \times P)$ *such that for all* $j \in O$ *and for all* $t \in T$,

(1) $\#\{p \in P \mid j \in F(p,t)\} = \#\{p \in P \mid j \in F(t,p)\} \leq 1$,

$V \subseteq tot(\{V_j \mid j \in O\})$, *where each* V_j *is a finite alphabet, is the* n*-dimensional vector alphabet of* vector labels *of* $\mathcal{N}$*, and*

$\ell : T \to V$ *is the* event labeling homomorphism *of* $\mathcal{N}$ *such that for all* $j \in O$ *and for all* $t \in T$,

(2) $proj_j(\ell(t)) \neq \lambda$ *if and only if* $j \in \bigcup_{p \in P}(F(p,t) \cup F(t,p))$. □

For each event t of $\mathcal{N}$ we denote the set $\bigcup_{p \in P}(F(p,t) \cup F(t,p))$ of *tokens used by* t by $\mathrm{use}(t)$.

A VLITN $\mathcal{N} = (P, T, O, F, V, \ell)$ is represented graphically by drawing its places as circles, its events as rectangles, and an arc from place (event) x to event (place) y whenever $F(x,y) \neq \varnothing$. Events are drawn together with their label and the arcs (x,y) are labeled with the elements constituting $F(x,y)$ (cf. Figure 7.6).

To define the dynamic behavior of a VLITN, we use the notion of *marking* to describe states defined by the locations of the individual tokens. These markings are (total) functions that assign a place to each of the tokens. Thus each token appears exactly once. A marking is graphically represented by drawing each token in the place in which it is present according to that marking.

At a certain marking of a VLITN, an event t is *enabled* (can occur) if in that marking each place p for which $F(p,t) \neq \varnothing$ contains at least the tokens specified in $F(p,t)$. When t consequently *fires* (occurs) all those tokens are removed and each place p for which $F(t,p) \neq \varnothing$ receives the tokens specified in $F(t,p)$.

Condition (1) in Definition 7.2.16 guarantees that every VLITN is *1-throughput*: for each event t, the tokens in $\bigcup_{p \in P} F(p,t)$ are exactly those in $\bigcup_{p \in P} F(t,p)$. Hence $\mathrm{use}(t) = \bigcup_{p \in P} F(p,t) = \bigcup_{p \in P} F(t,p)$. This condition furthermore guarantees that after an event has fired, no individual tokens have been added to or have disappeared from the VLITN, i.e. the resulting token distribution is again a marking of the VLITN.

Condition (2) in Definition 7.2.16 guarantees that every VLITN is *label consistent*: for each event t, the nonempty entries in its vector label correspond to the tokens actually used by t. Moreover, since Λ is not a vector letter, each event uses at least one token.

Definition 7.2.17. *Let* $\mathcal{N} = (P, T, O, F, V, \ell)$ *be a VLITN. Then*

(1) the set of all markings *of* $\mathcal{N}$ *is denoted by* $\mathbf{M}_{\mathcal{N}}$ *and is defined as* $\mathbf{M}_{\mathcal{N}} = \{\mu \mid \mu : O \to P\}$,

(2) an event t is enabled *at a marking $\mu \in \mathbf{M}_{\mathcal{N}}$, denoted by $\mu[t\rangle_{\mathcal{N}}$, if $F(p,t) \subseteq \{j \in O \mid \mu(j) = p\}$ for all $p \in P$,*

(3) an event t fires *from a marking $\mu \in \mathbf{M}_{\mathcal{N}}$ to a marking $\nu \in \mathbf{M}_{\mathcal{N}}$, denoted by $\mu[t\rangle_{\mathcal{N}}\nu$, if t is enabled at marking μ and ν is defined by $\nu(j) = p$ if $j \in F(t,p)$, for a $p \in P$, and $\nu(j) = \mu(j)$ otherwise, and*

(4) if $t_1, t_2, \ldots, t_m \in T$, with $m \geq 0$, and $\mu_0 \in \mathbf{M}_{\mathcal{N}}$ are such that there exist $\mu_1, \mu_2, \ldots, \mu_m \in \mathbf{M}_{\mathcal{N}}$ with $\mu_{i-1}[t_i\rangle_{\mathcal{N}}\mu_i$, for all $i \in [m]$, then $t_1 t_2 \cdots t_m$ is a firing sequence *of $\mathcal{N}$ starting from μ_0 (and leading to μ_m) denoted by $\mu_0[t_1 t_2 \cdots t_m\rangle_{\mathcal{N}}$ ($\mu_0[t_1 t_2 \cdots t_m\rangle_{\mathcal{N}}\mu_m$), and*

(5) if $t_1, t_2, \ldots \in T$ and $\mu_0 \in \mathbf{M}_{\mathcal{N}}$ are such that there exist $\mu_1, \mu_2, \ldots \in \mathbf{M}_{\mathcal{N}}$ with $\mu_{i-1}[t_i\rangle_{\mathcal{N}}\mu_i$, for all $i \geq 1$, then $t_1 t_2 \cdots$ is an infinite firing sequence *of $\mathcal{N}$ starting from μ_0 denoted by $\mu_0[t_1 t_2 \cdots\rangle_{\mathcal{N}}$.* □

Note that $\mu[\lambda\rangle_{\mathcal{N}}\mu$, for all $\mu \in \mathbf{M}_{\mathcal{N}}$. Note furthermore that all prefixes of an infinite firing sequence starting from a marking are (finite) firing sequences starting from that marking.

To define an ITNC we add initial and final markings to a VLITN. With each individual token we associate initial (final) local states and any combination of initial (final) places for each of the tokens is a possible initial (final) marking. We thus require the sets of initial and final markings to be *complete*. Formally, any marking $\mu : O \to P$ of a VLITN $\mathcal{N} = (P,T,O,F,V,\ell)$ can be viewed as a vector $\mu = \prod_{j \in O} \mu(j)$ of places. Hence $\mathrm{proj}_j(\mu) = \mu(j)$, for all $j \in O$. Each set $\mathcal{M} \subseteq \mathbf{M}_{\mathcal{N}}$ of markings of $\mathcal{N}$ satisfies the property that $\mathcal{M} \subseteq \prod_{j \in O} \mathrm{proj}_j(\mathcal{M})$. We say that $\mathcal{M}$ is *complete* if this inclusion is an equality: $\mathcal{M} = \prod_{j \in O} \mathrm{proj}_j(\mathcal{M})$. Complete sets of markings are thus characterized by the property that they can be specified by just giving for each token j its own set of places P_j. Then the intended set of markings is simply $\prod_{j \in O} P_j$.

Definition 7.2.18. *An n-*ITNC *is a construct $\mathcal{K} = (\mathcal{N}, \mathcal{M}_0, \mathcal{M}_f)$, where*

$\mathcal{N}$ is an n-VLITN,
$\mathcal{M}_0 \subseteq \mathbf{M}_{\mathcal{N}}$ is a complete set of initial markings *of $\mathcal{K}$, and*
$\mathcal{M}_f \subseteq \mathbf{M}_{\mathcal{N}}$ is a complete set of final markings *of $\mathcal{K}$.* □

For an n-ITNC $\mathcal{K} = (\mathcal{N}, \mathcal{M}_0, \mathcal{M}_f)$, the n-VLITN $\mathcal{N}$ is called the *underlying n-VLITN* of $\mathcal{K}$ and it is denoted by $\mathrm{und}(\mathcal{K})$.

The dynamic behavior of an ITNC $\mathcal{K}$ now is made up of firing sequences of its underlying VLITN $\mathrm{und}(\mathcal{K})$ that start in an initial marking of $\mathcal{K}$ and that end in a final marking of $\mathcal{K}$.

Definition 7.2.19. *Let $\mathcal{K} = (\mathcal{N}, \mathcal{M}_0, \mathcal{M}_f)$, with $\mathcal{N} = (P, T, O, F, V, \ell)$, be an ITNC. Then*

(1) the set of all firing sequences *of $\mathcal{K}$ is denoted by $\mathbf{FS}_\mathcal{K}$ and is defined as $\mathbf{FS}_\mathcal{K} = \{u \in T^* \mid \mu_0[u\rangle_\mathcal{N}\nu,\ \mu_0 \in \mathcal{M}_0,\ \nu \in \mathcal{M}_f\}$,*

(2) the set of all reachable markings *of $\mathcal{K}$ is denoted by $\mathbf{M}_\mathcal{K}$ and is defined as $\mathbf{M}_\mathcal{K} = \{\nu \in \mathbf{M}_\mathcal{N} \mid \mu_0[u\rangle_\mathcal{N}\nu,\ \mu_0 \in \mathcal{M}_0,\ u \in T^*\}$,*

(3) the behavior *of $\mathcal{K}$ is denoted by $\mathbf{B}_\mathcal{K}$ and is defined as $\mathbf{B}_\mathcal{K} = \{\ell(u) \in V^* \mid u \in \mathbf{FS}_\mathcal{K}\}$, and*

(4) the vector behavior *of $\mathcal{K}$ is denoted by $\mathbf{V}_\mathcal{K}$ and is defined as $\mathbf{V}_\mathcal{K} = coll(\mathbf{B}_\mathcal{K})$.* □

Note that the firing sequences of an ITNC are defined in terms of those of its underlying VLITN. When this VLITN $\mathcal{N}$ is clear from the context, then we may also write $\mu[u\rangle\nu$ rather than $\mu[u\rangle_\mathcal{N}\nu$, where μ, ν are markings and u is a sequence of events.

Note furthermore that any (successful) firing sequence of an ITNC leads from an initial marking to a final marking and is thus finite. Due to the finite number of tokens and places, each ITNC moreover has a finite set of reachable markings, i.e. a *finite state space*. Hence an ITNC is a finite-state system with a sequential behavior defined by its firing sequences.

Theorem 7.2.20. *Let $\mathcal{K}$ be an ITNC. Then*

$$\mathbf{FS}_\mathcal{K} \in \mathsf{REG}. \qquad \square$$

However, in contrast to a finite automaton an ITNC also allows concurrent behavior, as events may be enabled independent of one another. We call two events t and t' are *independent* if $\mathrm{use}(t) \cap \mathrm{use}(t') = \varnothing$, i.e. t and t' use different tokens. Consequently, whenever two independent events are simultaneously enabled at some marking of an ITNC, then they can fire in any order. This leads to an independence relation over the vector labels of the ITNC, similar to the independence relation used in *trace theory* (see, e.g. [Maz89] and [DR95]). In fact, as discussed in [KK97], (generalized) ITNCs are closely related to the well-known model of *finite asynchronous automata*, an automata model for (recognizable) trace languages (see, e.g., [Zie87] and [DR95]).

In the following example we illustrate the notion of an ITNC with an example derived from the Introduction of [Kee96].

Example 7.2.21. Two computers A and B share a single printer. A critical section is necessary to prohibit access to the printer for both computers at

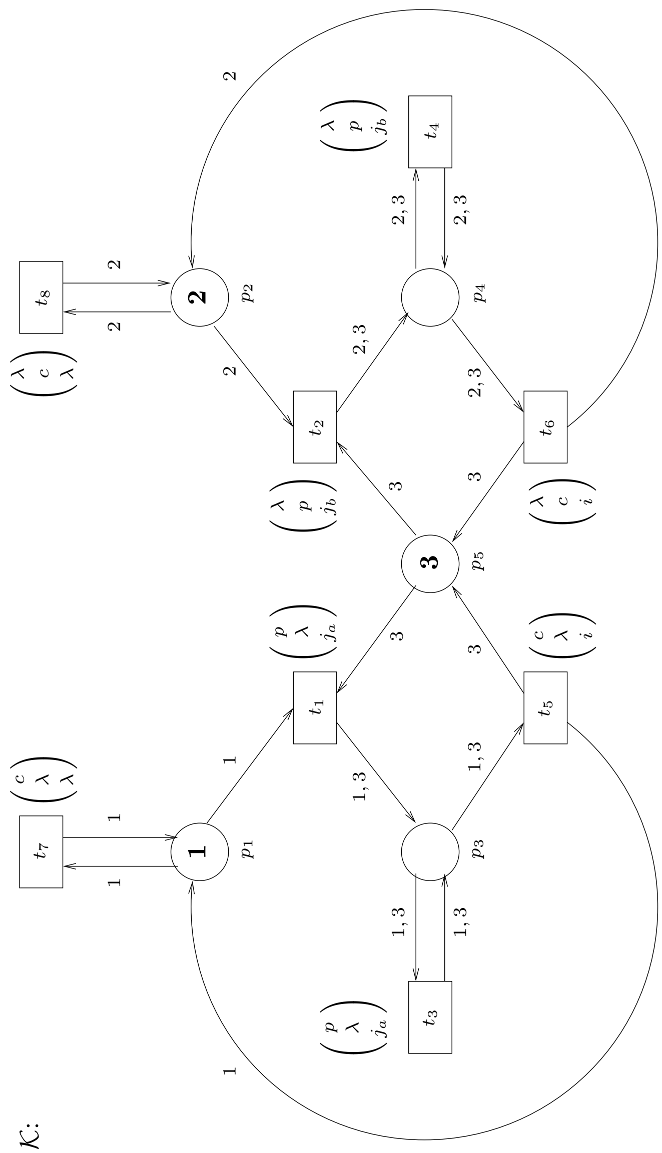


Fig. 7.6. 3-ITNC $\mathcal{K}$.

the same time. To model this with a 3-ITNC, we define the following actions for the computers and the printer. A computer can **c**alculate or **p**rint, in which case the printer indicates which computer is printing a **j**ob: j_a for computer A and j_b for computer B. Next to printing, the printer can also be **i**dle. These are all possible actions. However, some of these actions are synchronized. In this way, when the computer is printing, the printer indicates which one by synchronizing p with either j_a or j_b.

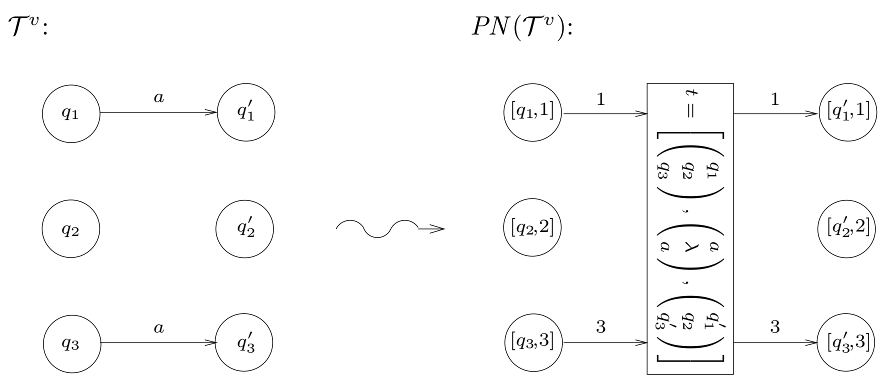
Let $\mathcal{K} = (\mathcal{N}, \mathcal{M}_0, \mathcal{M}_f)$, where $\mathcal{N} = (P, T, O, F, V, \ell)$, be a 3-ITNC. Its set of places P is $\{p_1, p_2, \ldots, p_5\}$, its set of events T is $\{t_1, t_2, \ldots, t_8\}$, its set of tokens O is [3] (represented as **1**, **2**, and **3** in Figure 7.6), its flow function F is as represented in Figure 7.6, its vector alphabet V consists of vector labels (c, λ, λ), (λ, c, λ), (p, λ, j_a), (λ, p, j_b), (c, λ, i), and (λ, c, i), its vector labeling homomorphism ℓ is as represented in Figure 7.6, its complete set of initial markings is $\{(p_1, p_2, p_5)\}$, and its complete set of final markings is $\{p_1, p_3\} \times \{p_2, p_4\} \times \{p_3, p_4, p_5\}$.

From the initial marking, both computers can (concurrently) calculate by firing t_7 and/or t_8, or one of them can print by firing t_1 or t_2. In case one of them starts printing, token **3** becomes unavailable for the other. The printing computer can then either continue printing (t_3 or t_4) or return to calculating (t_5 or t_6), in which case the printer becomes idle and token **3** becomes available for both computers again. Concurrently with the printing computer, the other can calculate (t_7 or t_8) but not print. These processes can be repeated and printing can be interchanged between both computers. Thus $t_8 t_7 t_1 t_3 t_8 t_5 t_7 t_2$ is a firing sequence of $\mathcal{K}$. Since $\{t_8, t_7\}$, $\{t_3, t_8\}$, and $\{t_7, t_2\}$ are pairs of events that can fire concurrently, also $u = t_7 t_8 t_1 t_8 t_3 t_5 t_2 t_7 \in \mathbf{FS}_{\mathcal{K}}$. As part of the behavior of $\mathcal{K}$ we thus have $\ell(u) = \begin{pmatrix} c \\ \lambda \\ \lambda \end{pmatrix}\begin{pmatrix} \lambda \\ c \\ \lambda \end{pmatrix}\begin{pmatrix} p \\ \lambda \\ j_a \end{pmatrix}\begin{pmatrix} \lambda \\ c \\ \lambda \end{pmatrix}\begin{pmatrix} p \\ \lambda \\ j_a \end{pmatrix}\begin{pmatrix} c \\ \lambda \\ i \end{pmatrix}\begin{pmatrix} \lambda \\ p \\ j_b \end{pmatrix}\begin{pmatrix} c \\ \lambda \\ \lambda \end{pmatrix}$ and $\mathrm{coll}\,(l(u)) = \begin{pmatrix} cppcc \\ ccp \\ j_a j_a i j_b \end{pmatrix}$ is part of the vector behavior of $\mathcal{K}$. □

From this example it also becomes clear that ITNCs are a particular kind of *state machine decomposable nets* (see, e.g., [BC92] and [JL92]). Each individual token uniquely determines a sequential subnet (a state machine or automaton with labeled transitions), every event represents a synchronization of transitions from these state machines, and the vector labeling such an event has a nonempty component for precisely those state machines involved in the synchronization. The initial (final) markings are any combination of initial (final) states of each of the state machines. Our translation of vector team automata into ITNCs follows this pattern by using the fact that also vector team automata are composed of automata connected by synchronizations. Recall that we have assumed already that $\mathcal{S}$ is a finite composable system.

Notation 22. *For the remainder of this chapter we moreover require that each of the component automata in $\mathcal{S}$ is finite, i.e. has a finite set of states and a finite alphabet.* □

To translate a given vector team automaton $\mathcal{T}^v$ into an ITNC that we will denote by $PN(\mathcal{T}^v)$, we use the construction sketched in Figure 7.7.



$$\left(\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}, \begin{pmatrix} a \\ \lambda \\ a \end{pmatrix}, \begin{pmatrix} q_1' \\ q_2 \\ q_3' \end{pmatrix}\right) \in \delta^v$$

$$\mu[t\rangle\nu,\ \ell(t) = (a, \lambda, a)$$
$$\mu(1) = [q_1, 1] \qquad \nu(1) = [q_1', 1]$$
$$\mu(3) = [q_3, 3] \qquad \nu(3) = [q_3', 3]$$

Fig. 7.7. Sketch of the construction of $PN(\mathcal{T}^v)$.

The individual tokens of $PN(\mathcal{T}^v)$ correspond to the component automata in $\mathcal{S}$. Hence the set of tokens O of $PN(\mathcal{T}^v)$ equals $\mathcal{I}$. The (local) states of the component automata correspond to places of $PN(\mathcal{T}^v)$. Since the Q_i, with $i \in \mathcal{I}$, are not necessarily pairwise disjoint, we distinguish them by indexing them. If state q belongs to both Q_i and Q_j, with $i, j \in \mathcal{I}$, then $PN(\mathcal{T}^v)$ will thus have places $[q, i]$ and $[q, j]$. The transitions of $\mathcal{T}^v$ will be the labeled events of $PN(\mathcal{T}^v)$. For a transition $(q, \underline{a}, q') \in \delta^v$, $PN(\mathcal{T}^v)$ will thus have the event $[q, \underline{a}, q']$ labelled by $\underline{a}$. Moreover, this event uses exactly those tokens which correspond to the component automata taking part in the synchronization $(q, \underline{a}, q')$. Let us call the set of indices of those component automata that participate in the execution of a vector action the *carrier* of that vector action. Hence, for a vector action $\underline{a}$, the *carrier* of $\underline{a}$ is denoted by carrier$(\underline{a})$ and is defined as carrier$(\underline{a}) = \{i \in \mathcal{I} \mid \mathrm{proj}_i(\underline{a}) \neq \lambda\}$. The flow function F of $PN(\mathcal{T}^v)$ now enforces that each event $[q, \underline{a}, q']$ uses exactly the tokens corresponding to the component automata taking part in $\underline{a}$ and, moreover, that these tokens are in the correct places (local states): whenever $\mathrm{proj}_i(\underline{a}) \neq \lambda$, then $F([\mathrm{proj}_i(q), i], [q, \underline{a}, q']) = F([q, \underline{a}, q'], [\mathrm{proj}_i(q'), i]) = \{i\}$, while for all other places p of $PN(\mathcal{T}^v)$, $i \notin F(p, [q, \underline{a}, q']) \cup F([q, \underline{a}, q'], p)$.

Let us say that a marking μ of $PN(\mathcal{T}^v)$ corresponds to a state q of $\mathcal{T}^v$ if μ puts token i in the place associated to the i-th element of q, i.e. $\mu = \prod_{i \in \mathcal{I}} [\mathrm{proj}_i(q), i]$. Observe that for every state q of $\mathcal{T}^v$ there is a unique corresponding marking, which we will denote by μ_q. Conversely, every marking μ of $PN(\mathcal{T}^v)$ corresponds to a state of $\mathcal{T}^v$ provided that each token i is assigned to a place indexed with i.

The initial markings of $PN(\mathcal{T}^v)$ will correspond to the initial states of $\mathcal{T}^v$, i.e. if $q \in I_i$, with $i \in \mathcal{I}$, then $[q, i]$ will be an initial place for token i. The set of initial markings then consists of all combinations of initial places for each of the tokens (which yields a complete set).

Vector team automaton $\mathcal{T}^v$ obviously has no final states, and we now have two options. Either we allow every marking of $PN(\mathcal{T}^v)$ as a final marking (which yields a complete set) or we allow as final markings all markings of $PN(\mathcal{T}^v)$ that correspond to a state of $\mathcal{T}^v$ (again yielding a complete set). Since we will see that the reachable markings of $PN(\mathcal{T}^v)$ all correspond to states of $\mathcal{T}^v$, which option we choose is irrelevant (cf. the remarks directly succeeding Lemma 7.2.30).

Formally, the construction of $PN(\mathcal{T}^v)$ is defined as follows.

Definition 7.2.22. *Let $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ be a vector team automaton over $\mathcal{S}$. Then $PN(\mathcal{T}^v)$ is defined as the construct $PN(\mathcal{T}^v) = (\mathcal{N}, \mathcal{M}_0, \mathcal{M}_f)$, where*

$\mathcal{N} = (P, T, O, F, V, \ell)$, *with*
- $P = \bigcup_{i \in \mathcal{I}} \{[q, i] \mid q \in Q_i\}$,
- $T = \{[q, \underline{a}, q'] \mid (q, \underline{a}, q') \in \delta^v\}$,
- $O = \mathcal{I}$,
- $F : (P \times T) \cup (T \times P) \to J$ *is defined by* $F([\mathit{proj}_i(q), i], [q, \underline{a}, q']) = F([q, \underline{a}, q'], [\mathit{proj}_i(q'), i]) = \{i\} \cap \mathit{carrier}(\underline{a})$,
- $V = \{\underline{a} \mid (q, \underline{a}, q') \in \delta^v$ *for some* $q, q' \in Q\}$, *and*
- $\ell : T \to V$ *is defined by* $\ell([q, \underline{a}, q']) = \underline{a}$,

$\mathcal{M}_0 = \{\mu_q \mid q \in I\}$, *and*
$\mathcal{M}_f = \{\mu_q \mid q \in Q\}$. □

In order to clarify the construction presented in this definition, we now apply it to two vector team automata from earlier examples.

Example 7.2.23. (Example 7.2.5 continued) In Figure 7.8, $PN(\mathcal{T}_2^v)$ as obtained by applying the construction of Definition 7.2.22 to vector team automaton $\mathcal{T}_2^v$, depicted in Figure 7.2, is given.

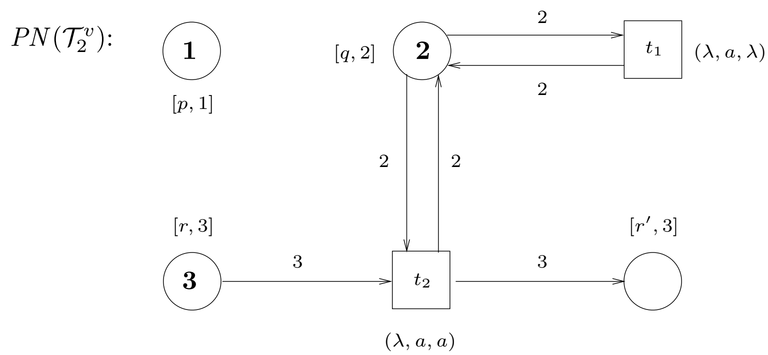


Fig. 7.8. $PN(\mathcal{T}_2^v)$.

This $PN(\mathcal{T}_2^v)$ has places $[p,1]$, $[q,2]$, $[r,3]$, and $[r',3]$, events $t_1 = [(p,q,r'),(\lambda,a,\lambda),(p,q,r')]$ and $t_2 = [(p,q,r),(\lambda,a,a),(p,q,r')]$, tokens **1**, **2**, and **3**, flow function F as represented in Figure 7.8, vector labels (λ,a,λ) and (λ,a,a) forming its vector alphabet, vector labeling homomorphism ℓ as represented in Figure 7.8, set of initial markings $\{([p,1],[q,2],[r,3])\}$, and set of final markings $\{([p,1],[q,2],[r,3]),([p,1],[q,2],[r',3])\}$. $\square$

Example 7.2.24. (Example 7.2.9 continued) We now apply the construction of Definition 7.2.22 to $\mathcal{T}^v_{\{1,2\}}$. This results in the 2-ITNC $PN(\mathcal{T}^v_{\{1,2\}}) = (\{[s_1,1],[s_2,2],[t_1,1],[t_2,2]\}, \{[(s_1,s_2),(b,b),(s_1,s_2)],[(s_1,s_2),(a,a),(t_1,t_2)], [(t_1,t_2),(a,a),(t_1,t_2)], [(t_1,t_2),(b,b),(s_1,s_2)]\}, [2], F, \{(a,a),(b,b)\}, \ell), \{([s_1,1],[s_2,2])\}, \{[s_1,1],[t_1,1]\} \times \{[s_2,2],[t_2,2]\})$, where F and ℓ are as represented in Figure 7.9.

Note that we have used some abbreviations in Figure 7.9, viz. $s_i = [s_i,i]$ and $t_i = [t_i,i]$, for $i \in [2]$, $u_1 = [(t_1,t_2),(b,b),(s_1,s_2)]$, $u_2 = [(s_1,s_2),(b,b),(s_1,s_2)]$, $u_3 = [(s_1,s_2),(a,a),(t_1,t_2)]$, and $u_4 = [(t_1,t_2),(a,a),(t_1,t_2)]$. $\square$

Since $PN(\mathcal{T}^v)$ is an ITNC, our goal of translating vector team automata into ITNCs has been achieved by the construction given in Definition 7.2.22.

Lemma 7.2.25. *Let $\mathcal{T}^v$ be a vector team automaton over $\mathcal{S}$. Then*

$PN(\mathcal{T}^v)$ is an ITNC.

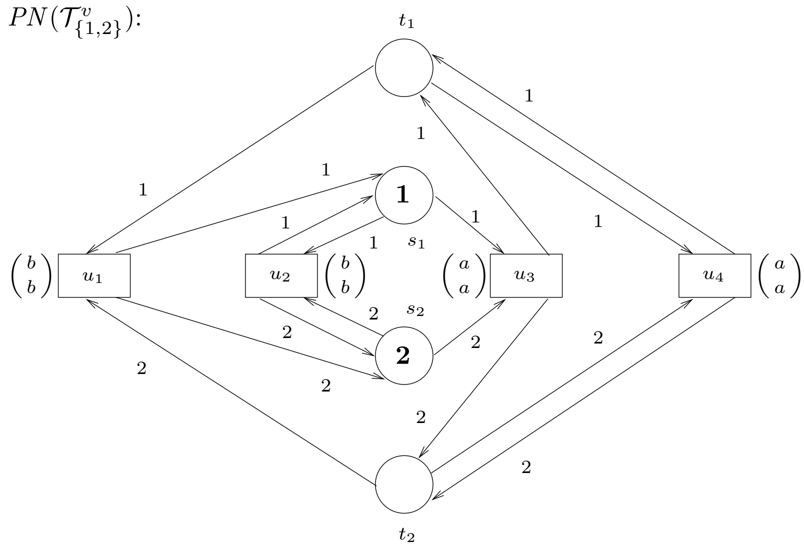


Fig. 7.9. ITNC $PN(\mathcal{T}^v_{\{1,2\}})$.

Proof. It is straightforward to verify that $PN(\mathcal{T}^v)$ as specified in Definition 7.2.22 satisfies the definition of an ITNC, in particular $\mathcal{N}$ is 1-throughput and label consistent, and $\mathcal{M}_0$ and $\mathcal{M}_f$ are complete sets of markings. □

Note that the set of ITNCs that can be obtained by applying the construction of Definition 7.2.22 to vector team automata forms a proper subclass of the complete set of ITNCs. It is not difficult to see that, e.g., the 3-ITNC $\mathcal{K}$ of Example 7.2.21 (depicted in Figure 7.6) cannot be obtained by applying the construction of Definition 7.2.22 to some vector team automaton. In fact, this example neatly illustrates three properties of ITNCs that are not inherited by its subclass obtained by applying the construction of Definition 7.2.22 to vector team automata. First, ITNCs may have pluriform synchronizations (cf. the Introduction). Secondly, ITNCs may have arcs labeled with subsets of tokens having a cardinality larger than one. Thirdly, ITNCs may have places that do not "belong" to specifically one component. This latter property should be understood in the sense that places of an ITNC may be part of the two (or more) different state machines that are determined by two (or more) different individual tokens.

At this point one might be inclined to conclude that the finite computations of a vector team automaton $\mathcal{T}^v$ are in a one-to-one correspondence with the firing sequences of the ITNC $PN(\mathcal{T}^v)$ such that both constructs exhibit the same (finitary) behavior. In the following two examples we however show that this in general is not the case.

Example 7.2.26. Let the component automata $\mathcal{C}_1 = (\{q_1, q_1'\}, (\varnothing, \{a\}, \varnothing), \{(q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{C}_2 = (\{q_2, q_2'\}, (\varnothing, \{b\}, \varnothing), \{(q_2, b, q_2')\}, \{q_2\})$ be as depicted in Figure 7.10.

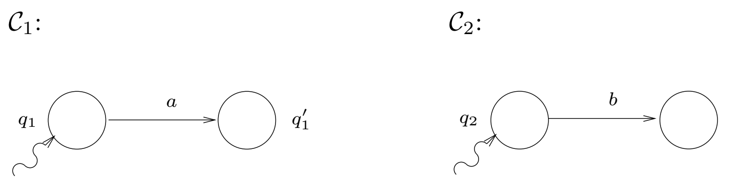


Fig. 7.10. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$.

Clearly, $\{\mathcal{C}_1, \mathcal{C}_2\}$ is a composable system. Consider the vector team automaton $\mathcal{T}_1^v = (Q, (\varnothing, \{a, b\}, \varnothing), \delta_1^v, \{(q_1, q_2)\})$, where $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$ and $\delta_1^v = \{((q_1, q_2), (a, \lambda), (q_1', q_2)), ((q_1', q_2), (\lambda, b), (q_1', q_2'))\}$, over $\{\mathcal{C}_1, \mathcal{C}_2\}$. It is depicted in Figure 7.11(a).

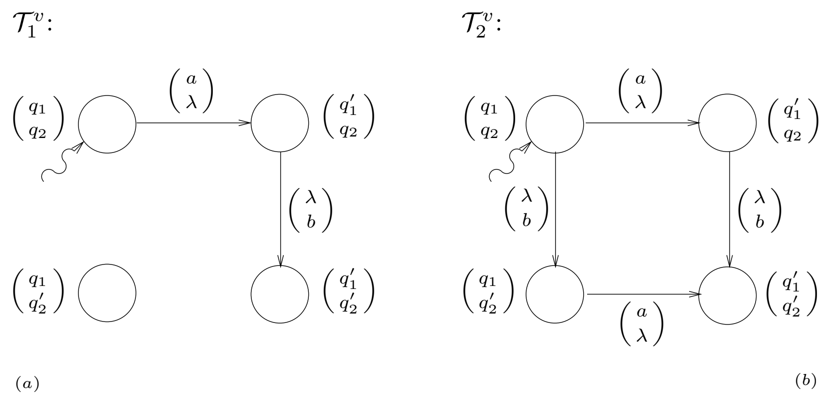


Fig. 7.11. Vector team automata $\mathcal{T}_1^v$ and $\mathcal{T}_2^v$.

In Figure 7.12 the ITNC $PN(\mathcal{T}_1^v) = (P, \{t_1, t_2\}, [2], F_1, V, \ell_1, \mathcal{M}_0, \mathcal{M}_f)$, with $P = \{[q_1, 1], [q_1', 1], [q_2, 2], [q_2', 2]\}$, $t_1 = [(q_1, q_2), (a, \lambda), (q_1', q_2)]$, $t_2 = [(q_1', q_2), (\lambda, b), (q_1', q_2')]$, $V = \{(a, \lambda), (\lambda, b)\}$, $\mathcal{M}_0 = \{([q_1, 1], [q_2, 2])\}$, and $\mathcal{M}_f = \{[q_1, 1], [q_1', 1]\} \times \{[q_2, 2], [q_2', 2]\}$, is depicted.

Since $\mathrm{use}\,(t_1) \cap \mathrm{use}\,(t_2) = \varnothing$, we know that t_1 and t_2 are independent events. Indeed, as both are enabled in the initial marking of $PN(\mathcal{T}_1^v)$ they

$PN(\mathcal{T}_1^v)$:

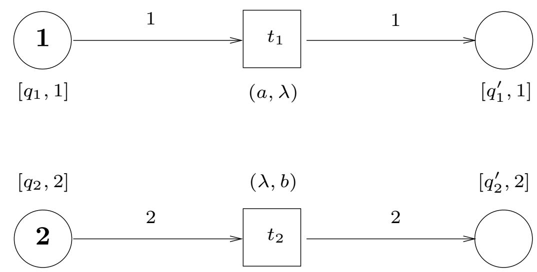


Fig. 7.12. ITNC $PN(\mathcal{T}_1^v)$.

can thus be fired in any order. In fact, it is easy to see that $\mathbf{B}_{PN(\mathcal{T}_1^v)} = \{\lambda, (a,\lambda), (\lambda,b), (a,\lambda)(\lambda,b), (\lambda,b)(a,\lambda)\}$. Note, however, that any nontrivial computation of $\mathcal{T}_1^v$ starts with the execution of the vector action (a,λ) through the transition $((q_1,q_2),(a,\lambda),(q_1',q_2))$ corresponding with t_1. In fact, we immediately see that $\mathbf{B}_{\mathcal{T}_1^v} = \{\lambda, (a,\lambda), (a,\lambda)(\lambda,b)\} \subset \mathbf{B}_{PN(\mathcal{T}_1^v)}$. □

This example shows that independent events that are enabled in the ITNC $PN(\mathcal{T}^v)$ obtained from the vector team automaton $\mathcal{T}^v$ can be fired in any order, even if the vector actions of their corresponding transitions in $\mathcal{T}^v$ cannot be executed in any order. More generally, as the following example shows, in an ITNC an enabled event can fire regardless of the whereabouts of any token it does not use.

Example 7.2.27. (Example 7.2.23 continued) Note that since token $\mathbf{2} \in \text{use}(t_1) \cap \text{use}(t_2)$, events t_1 and t_2 are not independent. Both events are enabled in the initial marking of $PN(\mathcal{T}_2^v)$ and they can clearly be fired in any order, i.e. $\{(\lambda,a,\lambda)(\lambda,a,a), (\lambda,a,a)(\lambda,a,\lambda)\} \subseteq \mathbf{B}_{PN(\mathcal{T}_2^v)}$. In fact, whether or not t_1 can fire can be decided regardless of the whereabouts of the tokens **1** and **3**. In $\mathcal{T}_2^v$, however, it is obvious that the vector action (λ,a,λ) can only be executed — through the transition $((p,q,r'),(\lambda,a,\lambda),(p,q,r'))$ corresponding to t_1 — when the third component automaton is in local state r', i.e. after the vector action (λ,a,a) has been executed through the transition $((p,q,r),(\lambda,a,a),(p,q,r'))$ corresponding to t_2. □

Summarizing we note that whereas ITNCs allow independent events to fire in any order, the vector transitions of vector team automata that involve disjoint sets of component automata in general cannot be executed in any order. As in ordinary team automata, transitions take place from selected combinations of local states from its component automata. The execution of an action in a given local state might thus depend on the states that other component automata are in. This is the concept coined *state sharing* in [EG02], as already mentioned in the beginning of this section. As shown in Example 7.2.27, ITNCs could be called *non-state-sharing*.

We now define *non-state-sharing* vector team automata as a class of vector team automata with the characteristic that whether or not a synchronization can take place only depends on the local states of the component automata actively involved in that synchronization.

Definition 7.2.28. *Let $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ be a vector team automaton over $\mathcal{S}$. Then*

> $\mathcal{T}^v$ *is* non-state-sharing *if whenever $(p, \underline{a}, p') \in \delta^v$, then for all $q \in Q$ such that for all $i \in carrier(\underline{a})$, $proj_i(q) = proj_i(p)$, we have $(q, \underline{a}, q') \in \delta^v$ with $proj_i(q') = proj_i(p')$ for all $i \in carrier(\underline{a})$, and $proj_i(q') = proj_i(q)$ for all other i.* □

As a consequence, synchronizations involving disjoint sets of component automata are independent and hence non-state-sharing vector team automata would allow a concurrent semantics.

Example 7.2.29. (Example 7.2.26 continued) Here we consider the vector team automaton $\mathcal{T}_2^v = (Q, (\varnothing, \{a, b\}, \varnothing), \delta_2^v, \{(q_1, q_2)\})$, in which $\delta_2^v = \delta_1^v \cup \{((q_1, q_2), (\lambda, b), (q_1, q_2')), ((q_1, q_2'), (a, \lambda), (q_1', q_2'))\}$, over $\{\mathcal{C}_1, \mathcal{C}_2\}$. It is depicted in Figure 7.11(b). Note that $\mathcal{T}_2^v$ is a non-state-sharing vector team automaton.

In Figure 7.13 the ITNC $PN(\mathcal{T}_2^v) = (P, \{t_1, t_2, t_3, t_4\}, [2], F_2, V, \ell_2, \mathcal{M}_0, \mathcal{M}_f)$, with $t_1 = [(q_1, q_2), (a, \lambda), (q_1', q_2)]$, $t_2 = [(q_1, q_2'), (a, \lambda), (q_1', q_2')]$, $t_3 = [(q_1, q_2), (\lambda, b), (q_1, q_2')]$, and $t_4 = [(q_1', q_2), (\lambda, b), (q_1', q_2')]$, is depicted.

We immediately see that $\mathbf{B}_{\mathcal{T}_2^v} = \mathbf{B}_{PN(\mathcal{T}_2^v)} = \mathbf{B}_{PN(\mathcal{T}_1^v)}$. □

We can now show that the finitary (vector) behavior of a non-state-sharing vector team automaton $\mathcal{T}^v$ equals the (vector) behavior of the ITNC $PN(\mathcal{T}^v)$. This is a direct consequence of the fact that every finite computation of $\mathcal{T}^v$ can be simulated by a firing sequence in $PN(\mathcal{T}^v)$, and vice versa.

To prove this latter statement we first observe that the occurrence of any transition $(p, \underline{a}, p')$ of $\mathcal{T}^v$ in a computation of $\mathcal{T}^v$ can be simulated by the event $[p, \underline{a}, p']$ firing from marking μ_p to marking $\mu_{p'}$. Here we do not need that $\mathcal{T}^v$ is a non-state-sharing vector team automaton. To prove the relationship the other way around, it would be convenient if $\mu[t\rangle\nu$ in $PN(\mathcal{T}^v)$, for some $t = [p, \underline{a}, p']$, would imply that $\mu = \mu_p$ and $\nu = \mu_{p'}$, where μ_p and $\mu_{p'}$ are the unique markings corresponding to p and p', respectively. This, however, is in general not the case. Even if $\mu = \mu_q$, for some $q \in Q$, then p and q may still differ. This is due to the property of ITNCs that for the occurrence of an

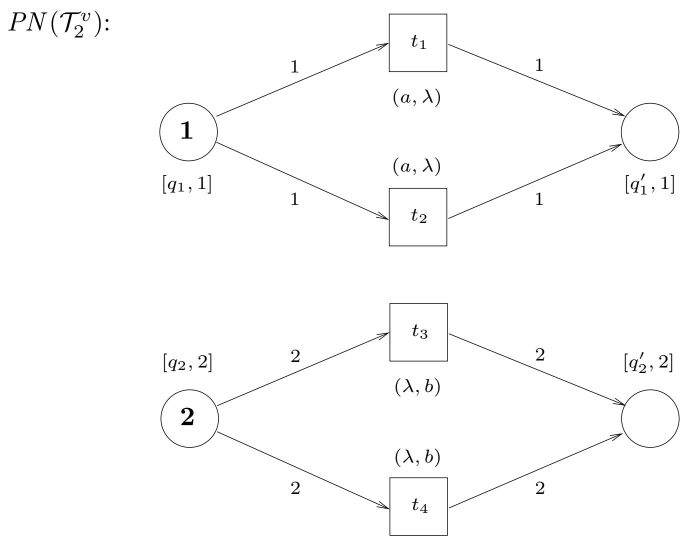


Fig. 7.13. ITNC $PN(\mathcal{T}_2^v)$.

event the whereabouts of the tokens it does not use is irrelevant. Since $\mathcal{T}^v$ is a non-state-sharing vector team automaton, we do know that $PN(\mathcal{T}^v)$ also has an event $t' = [q, \underline{a}, q']$ such that $\mathrm{proj}_i(q') = \mathrm{proj}_i(p')$, for all $i \in \mathrm{carrier}\,(\underline{a})$. The occurrence of t can now be simulated by $\mu_q[t'\rangle\nu$, with $\nu = \mu_{q'}$ in turn corresponding with the occurrence of the transition $(q, \underline{a}, q')$ in a computation of $\mathcal{T}^v$. The described situation is illustrated in Figure 7.14.

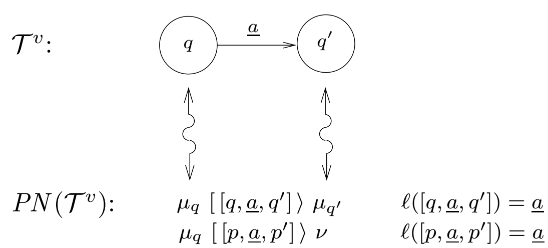


Fig. 7.14. Sketch of the idea underlying the simulation.

A more concrete example of the described situation occurs in the ITNC $PN(\mathcal{T}_2^v)$ depicted in Figure 7.13, where $([q_1, 1], [q_2', 2])[t_1\rangle([q_1', 1], [q_2', 2])$ with $t_1 = [(q_1, q_2), (a, \lambda), (q_1', q_2)]$. However, $PN(\mathcal{T}_2^v)$ also has the event $t_2 = [(q_1, q_2'), (a, \lambda), (q_1', q_2')]$ that can be used to simulate the occurrence of t_1 at $\mu_{(q_1, q_2')} = ([q_1, 1], [q_2', 2])$ and also leads to $\mu_{(q_1', q_2')} = ([q_1', 1], [q_2', 2])$. Whenever $([q_1, 1], [q_2', 2])[t_1\rangle([q_1', 1], [q_2', 2])$ appears in a firing sequence of $PN(\mathcal{T}_2^v)$ we may thus use the transition $((q_1, q_2'), (a, \lambda), (q_1', q_2'))$ in the corresponding computation of $\mathcal{T}^v$.

To avoid cumbersome descriptions, two transitions $(p, \underline{a}, p'), (q, \underline{a}, q') \in \Delta_a^v(\mathcal{S})$ are said to be *clones* whenever $\mathrm{proj}_i{}^{[2]}(p, p') = \mathrm{proj}_i{}^{[2]}(q, q')$, for all $i \in \mathrm{carrier}\,(\underline{a})$. In the vector team automaton $\mathcal{T}_2^v$ depicted in Figure 7.11(b), e.g., $((q_1, q_2), (a, \lambda), (q_1', q_2))$ and $((q_1, q_2'), (a, \lambda), (q_1', q_2'))$ are clones because $\mathrm{proj}_1{}^{[2]}((q_1, q_2), (q_1', q_2)) = (q_1, q_1') = \mathrm{proj}_1{}^{[2]}((q_1, q_2'), (q_1', q_2'))$. Since $\mathcal{T}^v$ is a non-state-sharing vector team automaton it follows that whenever $(p, \underline{a}, p') \in \delta^v$, then all clones of $(p, \underline{a}, p')$ are also transitions of $\mathcal{T}^v$.

Lemma 7.2.30. *Let $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ be a vector team automaton over $\mathcal{S}$ and let $PN(\mathcal{T}^v) = (\mathcal{N}, \mathcal{M}_0, \mathcal{M}_f)$, with $\mathcal{N} = (P, T, \mathcal{I}, F, V, \ell)$. Then*

(1) if $(p, \underline{a}, p') \in \delta^v$, then $\mu_p[[p, \underline{a}, p']\rangle\mu_{p'}$ in $PN(\mathcal{T}^v)$, and

(2) if $\mu_q[[p, \underline{a}, p']\rangle\nu$ in $PN(\mathcal{T}^v)$, with $p, p', q \in P$ and $\underline{a} \in tot(\{\Sigma_i \mid i \in \mathcal{I}\})$, then $\nu = \mu_{q'}$, where $q' \in Q$ is the unique state such that $(q, \underline{a}, q')$ and $(p, \underline{a}, p')$ are clones.

Proof. (1) Let $(p, \underline{a}, p') \in \delta^v$. Then $[p, \underline{a}, p']$ is an event of $PN(\mathcal{T}^v)$. That $[p, \underline{a}, p']$ is enabled at μ_p is easily seen as follows. By the construction of $PN(\mathcal{T}^v)$, in order to be able to fire $[p, \underline{a}, p']$ needs for all $i \in \mathrm{carrier}\,(\underline{a})$, token i in place $[\mathrm{proj}_i(p), i]$. This requirement is satisfied at μ_p because by definition $\mu_p(i) = [\mathrm{proj}_i(p), i]$, for all $i \in \mathrm{carrier}\,(\underline{a})$. Now let ν be the marking such that $\mu_p[[p, \underline{a}, p']\rangle\nu$ in $PN(\mathcal{T}^v)$. Then, again by the construction, $\nu(i) = [\mathrm{proj}_i(p'), i]$, for all $i \in \mathrm{carrier}\,(\underline{a})$, and $\nu(i) = \mu(i) = [\mathrm{proj}_i(p), i]$, for all $i \in \mathcal{I}$ for which $\mathrm{proj}_i(\underline{a}) = \lambda$. Hence $\nu = \mu_{p'}$.

(2) Let $\mu_q[[p, \underline{a}, p']\rangle\nu$ in $PN(\mathcal{T}^v)$, with $p, p', q \in P$ and $\underline{a} \in \mathrm{tot}\,(\{\Sigma_i \mid i \in \mathcal{I}\})$. Then for all $i \in \mathrm{carrier}\,(\underline{a})$, $\mu_q(i) = [\mathrm{proj}_i(p), i]$. Since by definition $\mu_q(i) = [\mathrm{proj}_i(q), i]$, for all $i \in \mathcal{I}$, it follows that $\mathrm{proj}_i(p) = \mathrm{proj}_i(q)$, for all $i \in \mathrm{carrier}\,(\underline{a})$. Recall that if $q' \in Q$ is the unique state such that $(q, \underline{a}, q')$ and (p, a, p') are clones, then $\mathrm{proj}_i(q') = \mathrm{proj}_i(p')$, for all $i \in \mathrm{carrier}\,(\underline{a})$ and $\mathrm{proj}_i(q') = \mathrm{proj}_i(q)$, for all $i \in \mathcal{I}$ such that $\mathrm{proj}_i(\underline{a}) = \lambda$. Given $\mu_q[[p, \underline{a}, p']\rangle\nu$ in $PN(\mathcal{T}^v)$, the definition of F implies that $\nu(i) = [\mathrm{proj}_i(p'), i]$, for all $i \in \mathrm{carrier}\,(\underline{a})$, and $\nu(i) = \mu_q(i) = [\mathrm{proj}_i(q), i]$, for all $i \in \mathcal{I}$ such that $\mathrm{proj}_i(\underline{a}) = \lambda$. Consequently, $\nu = \mu_{q'}$. □

Note that from Lemma 7.2.30(2) it immediately follows that $\mu_p[t\rangle\nu$ in $PN(\mathcal{T}^v)$ implies that there exists a state p' in $\mathcal{T}^v$ such that $\nu = \mu_{p'}$. Hence even when $\mathcal{T}^v$ is not a non-state-sharing vector team automaton, each reachable marking of $PN(\mathcal{T}^v)$ corresponds with a state of $\mathcal{T}^v$.

Theorem 7.2.31. *Let $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ be a non-state-sharing vector team automaton over $\mathcal{S}$ and let $PN(\mathcal{T}^v) = (\mathcal{N}, \mathcal{M}_0, \mathcal{M}_f)$, with $\mathcal{N} = (P, T, \mathcal{I}, F, V, \ell)$. Let $m \geq 1$ and let $(q_{j-1}, \underline{a}_j, q_j) \in \delta^v$, for all $1 \leq j \leq m$. Then*

> $[q_0, \underline{a}_1, q_1][q_1, \underline{a}_2, q_2] \cdots [q_{m-1}, \underline{a}_m, q_m] \in \mathbf{FS}_{PN(\mathcal{T}^v)}$ *if and only if for all $1 \leq j \leq m$, there exists a clone $(p_{j-1}, \underline{a}_j, p_j)$ of $(q_{j-1}, \underline{a}_j, q_j)$ such that $p_0\underline{a}_1 p_1 \underline{a}_2 p_2 \cdots p_{m-1}\underline{a}_m p_m \in \mathbf{C}_{\mathcal{T}^v}$.*

Proof. (If) Let $p_0\underline{a}_1 p_1 \underline{a}_2 p_2 \cdots p_{m-1}\underline{a}_m p_m \in \mathbf{C}_{\mathcal{T}^v}$ be such that $(p_{j-1}, \underline{a}, p_j)$ is a clone of $(q_{j-1}, \underline{a}_j, q_j)$, for all $1 \leq j \leq m$. Then the definition of $\mathbf{C}_{\mathcal{T}^v}$ implies that $p_0 \in I$. Furthermore, $(p_{j-1}, \underline{a}_j, p_j) \in \delta^v$, for all $1 \leq j \leq m$. From Lemma 7.2.30(1) we obtain that $\mu_{p_{j-1}}[[p_{j-1}, \underline{a}_j, p_j]\rangle \mu_{p_j}$ in $PN(\mathcal{T}^v)$, for all $1 \leq j \leq m$. Since $(p_{j-1}, \underline{a}_j, p_j)$ and $(q_{j-1}, \underline{a}_j, q_j)$ are clones for all $1 \leq j \leq m$, it follows immediately that for all places s of $PN(\mathcal{T}^v)$, $F(s, [p_{j-1}, \underline{a}_j, p_j]) = F(s, [q_{j-1}, \underline{a}_j, q_j])$ and $F([p_{j-1}, \underline{a}_j, p_j], s) = F([q_{j-1}, \underline{a}_j, q_j], s)$, for all $1 \leq j \leq m$. Thus we conclude from the above that $\mu_{p_{j-1}}[[q_{j-1}, \underline{a}_j, q_j]\rangle \mu_{p_j}$ in $PN(\mathcal{T}^v)$, for all $1 \leq j \leq m$. This implies that $\mu_{p_0}[[q_0, \underline{a}_1, q_1]\rangle \mu_{p_1}[[q_1, \underline{a}_2, q_2]\rangle \mu_{p_2} \cdots \mu_{p_{m-1}}[[q_{m-1}, \underline{a}_m, q_m]\rangle \mu_{p_m}$ in $PN(\mathcal{T}^v)$. As $p_0 \in I$, we have $\mu_{p_0} \in \mathcal{M}_0$. Moreover, μ_{p_m} is by definition a final marking of $PN(\mathcal{T}^v)$ and we may conclude that $[q_0, \underline{a}_1, q_1][q_1, \underline{a}_2, q_2] \cdots [q_{m-1}, \underline{a}_m, q_m] \in \mathbf{FS}_{PN(\mathcal{T}^v)}$.

(Only if) Let $[q_0, \underline{a}_1, q_1][q_1, \underline{a}_2, q_2] \cdots [q_{m-1}, \underline{a}_m, q_m] \in \mathbf{FS}_{PN(\mathcal{T}^v)}$ and let, for $0 \leq j \leq m$, μ_j be markings such that $\mu_{j-1}[[q_{j-1}, \underline{a}_j, q_j]\rangle \mu_j$ in $PN(\mathcal{T}^v)$, for all $1 \leq j \leq m$. Moreover, without loss of generality we may assume that μ_0 is an initial marking of $PN(\mathcal{T}^v)$. Let $p_0 \in I$ be the initial state of $\mathcal{T}^v$ such that $\mu_0 = \mu_{p_0}$. Combining Lemma 7.2.30(2) with the fact that $\mathcal{T}^v$ is a non-state-sharing vector team automaton now yields that both $\mu_{p_0}[[q_0, \underline{a}_1, q_1]\rangle \mu_{p_1}$ and $\mu_{p_0}[[p_0, \underline{a}_1, p_1]\rangle \mu_{p_1}$ in $PN(\mathcal{T}^v)$, with $(q_0, \underline{a}_1, q_1)$ and $(p_0, \underline{a}_1, p_1)$ being clones of each other. By repeatedly using this argumentation, we can conclude that for each $1 \leq j \leq m$, there exists a $p_j \in Q$ such that $\mu_{p_{j-1}}[[q_{j-1}, \underline{a}_j, q_j]\rangle \mu_{p_j}$ and $\mu_{p_{j-1}}[[p_{j-1}, \underline{a}_j, p_j]\rangle \mu_{p_j}$ in $PN(\mathcal{T}^v)$, with $(q_{j-1}, \underline{a}_j, q_j)$ and $(p_{j-1}, \underline{a}_j, p_j)$ being clones of each other. Consequently, $\mathcal{T}^v$ has transitions $(p_0, \underline{a}_1, p_1), (p_1, \underline{a}_2, p_2), \ldots, (p_{m-1}, \underline{a}_m, p_m)$ and since $p_0 \in I$, it follows that $p_0\underline{a}_1 p_1 \underline{a}_2 p_2 \cdots p_{m-1}\underline{a}_m p_m \in \mathbf{C}_{\mathcal{T}^v}$. □

The labeling of the events of $PN(\mathcal{T}^v)$ is in agreement with the vector labels of the corresponding transitions of the vector team automaton $\mathcal{T}^v$. Consequently, the (finitary) behavior of a vector team automaton $\mathcal{T}^v$ coincides with the behavior of $PN(\mathcal{T}^v)$ insofar it is based on nontrivial computations and nonempty firing sequences. In addition we observe that $\lambda \in \mathbf{FS}_{PN(\mathcal{T}^v)}$ if and only if the set of initial markings of $PN(\mathcal{T}^v) \neq \varnothing$ if and only if the set of initial states of $\mathcal{T}^v \neq \varnothing$ if and only if $\mathcal{T}^v$ has a trivial computation. We thus have the following result.

Theorem 7.2.32. *Let $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ be a non-state-sharing vector team automaton over $\mathcal{S}$. Then*

(1) $\mathbf{B}_{PN(\mathcal{T}^v)} = \mathbf{B}_{\mathcal{T}^v}$ *and*

(2) $\mathbf{V}_{PN(\mathcal{T}^v)} = \mathbf{V}_{\mathcal{T}^v}$. □

Recall that we have equiped ITNCs with final markings and thus with finitary behavior. By ignoring these final markings and by using the fact that we have seen that every prefix of an infinite firing sequence starting from an initial marking is a (finite) firing sequence starting from that marking, we could define the infinitary behavior of an ITNC as obtained from infinite firing sequences, which are limits of finite firing sequences. As we have seen in Theorem 7.2.31, the finite computations of a non-state-sharing vector team automaton $\mathcal{T}^v$ correspond to the finite firing sequences of the ITNC $PN(\mathcal{T}^v)$. Hence we can use the fact that the infinitary behavior of $\mathcal{T}^v$ can be fully determined by its finite computations (cf. Theorems 3.1.6 and 3.1.10) to conclude that $\mathcal{T}^v$ and $PN(\mathcal{T}^v)$ exhibit also the same infinitary behavior.

We now conclude this chapter with an observation relating the ITNC obtained by applying the construction of Definition 7.2.22 to the subteam determined by J of a vector team automaton $\mathcal{T}^v$ with a rather straightforward type of *subnet* of the ITNC $PN(\mathcal{T}^v)$. Mirroring the way we defined subteams of (vector) team automata, a subnet of an ITNC can be obtained by focusing on a subset of its set of individual tokens.

Recall that the restriction of a function $f : A \to A'$ to a subset C of its domain A is denoted by $f \upharpoonright C$ and is defined as the function $C \to A'$ defined by $(f \upharpoonright C)(c) = f(c)$, for all $c \in C$.

Definition 7.2.33. *Let $\mathcal{T}^v$ be a vector team automaton over $\mathcal{S}$ and let $\mathcal{K} = (\mathcal{N}, \mathcal{M}_0, \mathcal{M}_f)$, with $\mathcal{N} = (P, T, \mathcal{I}, F, V, \ell)$, be the ITNC $PN(\mathcal{T}^v)$. Let $J \subseteq \mathcal{I}$. Then the* subnet of $\mathcal{K}$ determined by J *is denoted by $SUB_J(\mathcal{K})$ and is defined as $SUB_J(\mathcal{K}) = (\mathcal{N}_J, (\mathcal{M}_0)_J, (\mathcal{M}_f)_J)$, where*

$\mathcal{N}_J = (P_J, T_J, J, F_J, V_J, \ell_J)$, *in which*
$P_J = \{[q, j] \mid q \in Q_j,\ j \in J\}$,
$T_J = \{[proj_J(q), proj_J(\underline{a}), proj_J(q')] \mid [q, \underline{a}, q'] \in T$ *for some* $q, q' \in Q$ *and* $J \cap carrier(\underline{a}) \neq \varnothing\}$,
$F_J : (P_J \times T_J) \cup (T_J \times P_J) \to J$ *is defined by* $F_J([proj_i(q), i], [proj_J(q), proj_J(\underline{a}), proj_J(q')]) = F_J([proj_J(q), proj_J(\underline{a}), proj_J(q')], [proj_i(q'), i]) = \{i\} \cap carrier(\underline{a})$,

$$V_J = \{\underline{b} \mid [p, \underline{b}, p'] \in T_J \textit{ for some } p, p' \in \textit{proj}_J(Q)\}, \textit{ and}$$
$$\ell_J : T_J \to V_J \textit{ is defined by } \ell_J([p, \underline{b}, p']) = \underline{b},$$
$$(\mathcal{M}_0)_J = \{\mu \restriction J \mid \mu \in \mathcal{M}_0\}, \textit{ and}$$
$$(\mathcal{M}_f)_J = \{\nu \restriction J \mid \nu \in \mathcal{M}_f\}. \qquad \square$$

Note that a subnet $SUB_J(PN(\mathcal{T}^v))$ of an ITNC $PN(\mathcal{T}^v)$ — both as specified in Definition 7.2.33 — is not simply defined by a local operation on the elements of the ITNC, but rather by a (syntactical) operation that refers to the transitions of $\mathcal{T}^v$ underlying the events of $PN(\mathcal{T}^v)$ and which is based on the actual participation of the component automata forming the subteam $SUB_J(\mathcal{T}^v)$. As a consequence, each event t of the subnet comprises all events $[q, \underline{a}, q']$ in the full net such that $[\mathrm{proj}_J(q), \mathrm{proj}_J(\underline{a}), \mathrm{proj}_J(q')] = t$. By the definition of the flow function F, whenever two events $[q, \underline{a}, q'], [p, \underline{a}, p'] \in T$ are such that $[\mathrm{proj}_J(q), \mathrm{proj}_J(\underline{a}), \mathrm{proj}_J(q')] = [\mathrm{proj}_J(p), \mathrm{proj}_J(\underline{a}), \mathrm{proj}_J(p')]$, then their neighborhoods when restricted to arcs with labels from J are the same. The definition of the flow function F_J then guarantees that also the labeled arcs connecting t with places $[p, j]$ in $SUB_J(PN(\mathcal{T}^v))$ correspond to the labeled arcs connecting the original events $[q, \underline{a}, q']$ with $[p, j]$ in $PN(\mathcal{T}^v)$.

Since the set of places P_J of $SUB_J(PN(\mathcal{T}^v))$ is a subset of P, the set of places of $PN(\mathcal{T}^v)$, the flow function F_J may be viewed as a restriction of the flow function F to $(P_J \times T_J) \cup (T_J \times P_J)$ once T has been transformed into T_J. Since V_J and ℓ_J agree with V and ℓ after projection, respectively, and since $(\mathcal{M}_0)_J$ and $(\mathcal{M}_f)_J$ are the restrictions of $\mathcal{M}_0$ and $\mathcal{M}_f$ to J, respectively, it is appropriate to refer to $SUB_J(PN(\mathcal{T}^v))$ as a subnet of the ITNC $PN(\mathcal{T}^v)$.

Example 7.2.34. (Example 7.2.29 continued) In Figure 7.15 the subnet determined by $\{1\}$ of $PN(\mathcal{T}_2^v)$ is depicted.

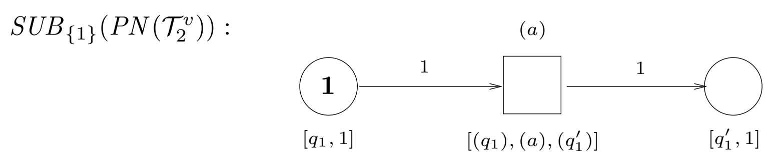


Fig. 7.15. ITNC $SUB_{\{1\}}(PN(\mathcal{T}_2^v))$.

We immediately see that the events $t_1 = [(q_1, q_2), (a, \lambda), (q_1', q_2)]$ and $t_2 = [(q_1, q_2'), (a, \lambda), (q_1', q_2')]$ of $PN(\mathcal{T}_2^v)$ have resulted in one and the same event $[\mathrm{proj}_1((q_1, q_2)), \mathrm{proj}_1((a, \lambda)), \mathrm{proj}_1((q_1', q_2))] = [\mathrm{proj}_1((q_1, q_2')), \mathrm{proj}_1((a, \lambda)), \mathrm{proj}_1((q_1', q_2'))] = [(q_1), (a), (q_1')]$ in $SUB_{\{1\}}(PN(\mathcal{T}_2^v))$. This reflects the fact that the dynamics of $SUB_{\{1\}}(PN(\mathcal{T}_2^v))$ is based on the actual participation of $\mathcal{C}_1$ — as the only component automaton forming $SUB_{\{1\}}(\mathcal{T}_2^v)$ depicted in Figure 7.16 — in the transitions of $\mathcal{T}_2^v$ that underlie the events of $PN(\mathcal{T}_2^v)$.

$SUB_{\{1\}}(\mathcal{T}_2^v)$:

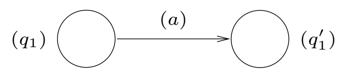


Fig. 7.16. Subteam $SUB_{\{1\}}(\mathcal{T}_2^v)$.

Analogously, we note that the transitions $((q_1, q_2), (a, \lambda), (q_1', q_2))$ and $((q_1, q_2'), (a, \lambda), (q_1', q_2'))$ of $\mathcal{T}_2^v$ have resulted in one and the same transition $((q_1), (a), (q_1'))$ in $SUB_{\{1\}}(\mathcal{T}_2^v)$. □

It is not hard to see that a subnet of an ITNC $PN(\mathcal{T}^v)$ obtained by applying the construction of Definition 7.2.22 to a vector team automaton $\mathcal{T}^v$ is indeed an ITNC.

Theorem 7.2.35. *Let $\mathcal{T}^v$ be a vector team automaton over $\mathcal{S}$ and let $\mathcal{K} = PN(\mathcal{T}^v)$. Let $J \subseteq \mathcal{I}$. Then*

$SUB_J(\mathcal{K})$ is an ITNC.

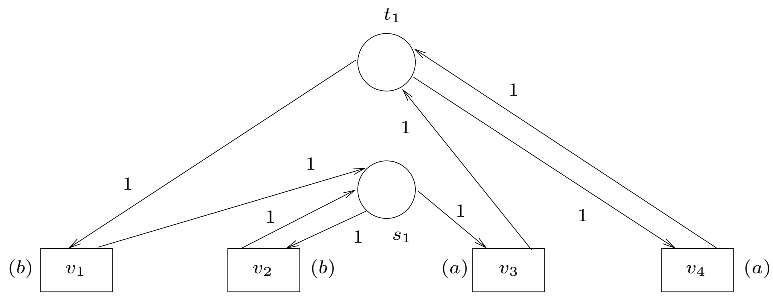
Proof. It is straightforward to verify that the subnet $SUB_J(\mathcal{K})$ as specified in Definition 7.2.33 satisfies the definition of an ITNC, in particular $\mathcal{N}_J$ is 1-throughput and label-consistent, and $(\mathcal{M}_0)_J$ and $(\mathcal{M}_f)_J$ are both complete sets of markings. □

Example 7.2.36. (Example 7.2.24 continued) In Figure 7.17, the underlying VLITNs $\mathrm{und}(SUB_{\{1\}}(PN(\mathcal{T}^v_{\{1,2\}})))$ and $\mathrm{und}(SUB_{\{2\}}(PN(\mathcal{T}^v_{\{1,2\}})))$ of subnets $SUB_{\{1\}}(PN(\mathcal{T}^v_{\{1,2\}}))$ and $SUB_{\{2\}}(PN(\mathcal{T}^v_{\{1,2\}}))$, respectively, are depicted.

Note that we have used some abbreviations in Figure 7.17, viz. $s_i = [s_i, i]$ and $t_i = [t_i, i]$, for $i \in [2]$, $v_1 = [(t_1), (b), (s_1)]$, $v_2 = [(s_1), (b), (s_1)]$, $v_3 = [(s_1), (a), (t_1)]$, $v_4 = [(t_1), (a), (t_1)]$, $w_1 = [(t_2), (b), (s_2)]$, $w_2 = [(s_2), (b), (s_2)]$, $w_3 = [(s_2), (a), (t_2)]$, and $w_4 = [(t_2), (a), (t_2)]$

Subnet $SUB_{\{1\}}(PN(\mathcal{T}^v_{\{1,2\}}))$ has $\{[s_1, 1]\}$ as its set of initial markings and $\{[s_1, 1], [t_1, 1]\}$ as its set of final markings. Subnet $SUB_{\{2\}}(PN(\mathcal{T}^v_{\{1,2\}}))$ has set of initial markings $\{[s_2, 2]\}$ and set of final markings $\{[s_2, 2], [t_2, 2]\}$. □

Definition 7.2.33 provides us with a notion of subnet for those ITNCs that result from applying the construction of Definition 7.2.22 to a vector team automaton. This definition of a subnet explicitly uses the relation to the original vector team automaton and is based on the participation of those automata forming the subteam in its transitions (and hence in the events of und($SUB_{\{1\}}(PN(\mathcal{T}^v_{\{1,2\}}))$):



und($SUB_{\{2\}}(PN(\mathcal{T}^v_{\{1,2\}}))$):

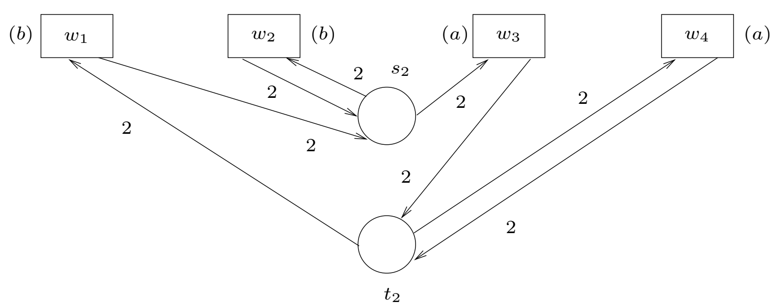


Fig. 7.17. VLITNs und($SUB_{\{1\}}(PN(\mathcal{T}^v_{\{1,2\}}))$) and und($SUB_{\{2\}}(PN(\mathcal{T}^v_{\{1,2\}}))$).

the subnet). As the following theorem shows, this notion of a subnet is correct in the sense that first constructing a net for a given vector team automaton and then extracting a subnet always yields the ITNC that results when first restricting the vector team automaton to a subteam and then constructing its net.

Theorem 7.2.37. *Let $\mathcal{T}^v = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta^v, I)$ be a vector team automaton over $\mathcal{S}$ and let $J \subseteq \mathcal{I}$. Then*

$$SUB_J(PN(\mathcal{T}^v)) = PN(SUB_J(\mathcal{T}^v)).$$

Proof. By inspecting Definitions 7.2.22 and 7.2.33 on the one hand and Definitions 7.2.3 and 7.2.22 on the other hand, we show element-wise that $SUB_J(PN(\mathcal{T}^v)) = PN(SUB_J(\mathcal{T}^v))$. To this aim, let $SUB_J(PN(\mathcal{T}^v)) = (\mathcal{N}_1, (\mathcal{M}_0)_1, (\mathcal{M}_f)_1)$, with $\mathcal{N}_1 = (P_1, T_1, J, F_1, V_1, \ell_1)$, and let $PN(SUB_J(\mathcal{T}^v)) = (\mathcal{N}_2, (\mathcal{M}_0)_2, (\mathcal{M}_f)_2)$, with $\mathcal{N}_2 = (P_2, T_2, J, F_2, V_2, \ell_2)$.

It is immediate that $P_1 = P_2 = \bigcup_{j \in J}\{[q,j] \mid q \in Q_j\}$, i.e. the set of places of $SUB_J(PN(\mathcal{T}^v))$ and that of $PN(SUB_J(\mathcal{T}^v))$ are identical.

It is also clear that $T_1 = \{[\mathrm{proj}_J(q), \mathrm{proj}_J(\underline{a}), \mathrm{proj}_J(q')] \mid [q, \underline{a}, q'] \in T$ for some $q, q' \in Q$ and $J \cap \mathrm{carrier}(\underline{a}) \neq \varnothing\} = \{[\mathrm{proj}_J(q), \mathrm{proj}_J(\underline{a}), \mathrm{proj}_J(q')] \mid (q, \underline{a}, q') \in \delta^v$ and $\mathrm{proj}_J(\underline{a}) \neq \Lambda\} = \{[\mathrm{proj}_J(q), \mathrm{proj}_J(\underline{a}), \mathrm{proj}_J(q')] \mid (\mathrm{proj}_J(q), \mathrm{proj}_J(\underline{a}), \mathrm{proj}_J(q')) \in \Delta_a^v(\{\mathcal{C}_j \mid j \in J\})$ and $(q, \underline{a}, q') \in \delta^v\} = T_2$, i.e. the set of events of $SUB_J(PN(\mathcal{T}^v))$ and that of $PN(SUB_J(\mathcal{T}^v))$ are identical.

Let $p \in P_1 = P_2$ and let $t \in T_1 = T_2$. Let $i \in J$. Then $i \in F_1(p,t)$ if and only if there exist $q, q' \in Q$ and an $\underline{a} \in V_J$ such that $t = [\mathrm{proj}_J(q), \underline{a}, \mathrm{proj}_J(q')]$ and $i \in \mathrm{carrier}(\underline{a})$, and moreover $p = [\mathrm{proj}_i(q), i]$. This is equivalent with $i \in F_2(p,t)$. We thus conclude that $F_1(p,t) = F_2(p,t)$. Likewise, $F_1(t,p) = F_2(t,p)$ and hence the flow function of $SUB_J(PN(\mathcal{T}^v))$ and that of $PN(SUB_J(\mathcal{T}^v))$ are identical.

Since $T_1 = T_2 = \{[q, \underline{a}, q'] \mid (q, \underline{a}, q') \in \delta_J^v\}$, it follows immediately that $V_1 = \{\underline{b} \mid [p, \underline{b}, p'] \in T_1$ for some $p, p' \in \mathrm{proj}_J(Q)\} = \{\underline{b} \mid (p, \underline{b}, p') \in \delta_J^v$ for some $p, p' \in Q_J\} = V_2$ and that $\ell_1([r, \underline{c}, r']) = \ell_2([r, \underline{c}, r']) = \underline{c} \in V_1 = V_2$, for all $[r, \underline{c}, r'] \in T_1 = T_2$, i.e. the vector alphabet of vector labels and the vector labeling homomorphism of $SUB_J(PN(\mathcal{T}^v))$ and those of $PN(SUB_J(\mathcal{T}^v))$, respectively, are identical.

Finally, we immediately see that $(\mathcal{M}_0)_1 = \{\mu_q \upharpoonright J \mid q \in I\} = \{\mu_{\mathrm{proj}_J(q)} \mid \mathrm{proj}_J(q) \in I_J\} = (\mathcal{M}_0)_2$ and $(\mathcal{M}_f)_1 = \{\mu_q \upharpoonright J \mid q \in Q\} = \{\mu_{\mathrm{proj}_J(q)} \mid \mathrm{proj}_J(q) \in Q_J\} = (\mathcal{M}_f)_2$, i.e. $SUB_J(PN(\mathcal{T}^v))$ and $PN(SUB_J(\mathcal{T}^v))$ have the same set of initial markings as well as the same set of final markings.

Hence we have proven that $SUB_J(PN(\mathcal{T}^v)) = PN(SUB_J(\mathcal{T}^v))$. □

Example 7.2.38. (Example 7.2.34 continued) From Theorem 7.2.37 we now conclude that the ITNC $SUB_{\{1\}}(PN(\mathcal{T}_2^v))$ depicted in Figure 7.15 is identical to the ITNC obtained by applying the construction of Definition 7.2.22 to $SUB_{\{1\}}(\mathcal{T}_2^v)$, i.e. $SUB_{\{1\}}(PN(\mathcal{T}_2^v)) = PN(SUB_{\{1\}}(\mathcal{T}_2^v))$. □

Example 7.2.39. (Example 7.2.36 continued) From Theorem 7.2.37 we now conclude that the underlying VLITNs $\mathrm{und}(PN(SUB_{\{1\}}(\mathcal{T}_{\{1,2\}}^v)))$ and $\mathrm{und}(PN(SUB_{\{2\}}(\mathcal{T}_{\{1,2\}}^v)))$ of the ITNCs $PN(SUB_{\{1\}}(\mathcal{T}_{\{1,2\}}^v))$ and $PN(SUB_{\{2\}}(\mathcal{T}_{\{1,2\}}^v))$, respectively, are depicted in Figure 7.17 (obviously with the abbreviations spelled out in Example 7.2.36). □

7.2.5 Conclusion

In this section we have introduced vector team automata by switching from (team) actions to vectors of (component) actions. By inspecting the vector actions of vector team automata we were able to obtain precise information as to which component automata participate in which synchronizations. We have used this knowledge to formalize the notions of *free*, *ai*, and *si* actions in vector team automata based on information unavailable in ordinary team automata.

This transfer from actions to vector actions moreover made explicit the concurrency inherent to team automata, which allowed us to view (vector) team automata as VCCSs. In particular, we were able to relate a subclass of (vector) team automata to ITNCs, a model of vector labeled Petri nets developed within the VCCS framework. Though related, a number of important differences remain between both models, especially concerning the type of synchronizations that can be modeled. Whereas all vector letters of vector team automata are uniform, this does not hold in case of ITNCs. In this respect, ITNCs thus allow the modeling of more types of synchronization than (vector) team automata do. However, ITNCs are not concerned with the distinction of actions into input, output, and internal actions, which we have seen to be a crucial modeling feature of team automata. Furthermore, ITNCs are finite-state systems, whereas (vector) team automata may have an infinite number of states (and are thus more powerful from a computational point of view).

Finally, vector team automata — like team automata (cf. Section 5.2) but contrary to ITNCs — allow the construction of hierarchical systems in a natural way, viz. by iteratively composing vector team automata over vector team automata. Theorem 7.2.32 moreover provides a relation between finite non-state-sharing vector team automata and the subclass of ITNCs obtained by applying the construction of Definition 7.2.22 to finite vector team automata. For this particular subclass of ITNCs, Theorems 7.2.32 and 7.2.37 thus hint at a way around this limitation of ITNCs. However, since we have no characterization of this particular subclass of ITNCs, in Figure 7.18 no more than a hint towards iteratively composing a subclass of ITNCs is sketched.

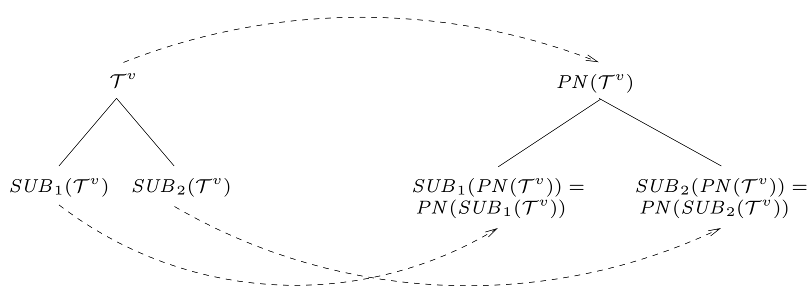


Fig. 7.18. Sketch of iteratively composing ITNCs.

Here $\mathcal{T}^v$ is a nontrivial finite vector team automaton with subteams $SUB_1(\mathcal{T}^v)$ and $SUB_2(\mathcal{T}^v)$. From Theorem 7.2.32 we know that $PN(\mathcal{T}^v)$, $PN(SUB_1(\mathcal{T}^v))$ and $PN(SUB_2(\mathcal{T}^v))$ have the same (vector) behavior as $\mathcal{T}^v$, $SUB_1(\mathcal{T}^v)$ and $SUB_2(\mathcal{T}^v)$, respectively. Moreover, from Theorem 7.2.37 we know that $SUB_1(PN(\mathcal{T}^v)) = PN(SUB_1(\mathcal{T}^v))$ and $SUB_2(PN(\mathcal{T}^v)) = PN(SUB_2(\mathcal{T}^v))$. Hence $PN(\mathcal{T}^v)$ might be seen as an ITNC iteratively composed over the ITNCs $PN(SUB_1(\mathcal{T}^v))$ and $PN(SUB_2(\mathcal{T}^v))$.

8. Applying Team Automata

In this chapter we give an impression of how team automata may be applied. We do this by presenting — in a varying degree of detail — three examples, each of which shows the usefulness of team automata in the early phases of system design. Additionally, we would like to mention that in [BLP03] we have initiated the use of team automata for the security analysis of multicast and broadcast communication. To this aim, team automata were used to model an instance of a particular stream signature protocol, while a well-established theory for defining and verifying a variety of security properties was reformulated in terms of team automata.

First we show — at a high level of abstraction — how to model a specific groupware architecture by team automata. To this aim we explain how team automata can be used as building blocks by internalizing certain external actions in order to prohibit their further use on a higher level of the construction (without changing the behavior of course).

Secondly, we show how team automata can be employed to model collaboration between teams of developers engaged in the development of models of complex (software) systems. This thus provides an example of using team automata for modeling interaction between humans. However, we still abstract from any social aspects and informal unstructured activity between humans. The team automata model solely the collaboration between humans.

Thirdly, we present a more detailed example demonstrating the potential of team automata for capturing information security and protection structures, and critical coordinations between these structures. On the basis of a spatial access metaphor, various known access control strategies are formally specified in terms of synchronizations in team automata. In [BB03] we have initiated an attempt to validate some of the resulting specifications with the model checker SPIN (see, e.g., [Hol91], [Hol97], and [Hol03]).

8.1 Groupware Architectures

In this section we show how team automata can be employed to model groupware architectures. To this aim we first introduce some notions and operations that are particularly useful when team automata are used for component-based system design. Consequently we use these operations to model a specific groupware architecture.

Notation 23. *Within this section we once again assume a fixed, but arbitrary and possibly infinite index set $\mathcal{I} \subseteq \mathbb{N}$, which we will use to index the component automata involved. For each $i \in \mathcal{I}$, we let $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$ be a fixed component automaton and we use $\Sigma_{i,ext}$ to denote its set of external actions $\Sigma_{i,inp} \cup \Sigma_{i,out}$. Moreover, we once again let $\mathcal{S} = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$ be a fixed composable system and we let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a fixed team automaton over $\mathcal{S}$. Furthermore, we use Σ to denote the set of actions $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$ and we use Σ_{ext} to denote the set of external actions $\Sigma_{inp} \cup \Sigma_{out}$ of any team automaton over $\mathcal{S}$. Recall that $\mathcal{I} \subseteq \mathbb{N}$ implies that $\mathcal{I}$ is ordered by the usual $\leq$ relation on $\mathbb{N}$, thus inducing an ordering on $\mathcal{S}$, and that the $\mathcal{C}_i$ are not necessarily different.* □

8.1.1 Team Automata as Architectural Building Blocks

As we have seen, a team automaton over a composable system is itself a component automaton that can be used in further constructions of team automata. Team automata can thus be used as building blocks. Before a team automaton is used as a building block, however, it may be necessary to internalize certain external actions in order to prohibit their further use on a higher level of the construction. The operation of hiding makes certain external actions of a component automaton invisible to other component automata by turning these external actions into internal actions. This operation has also been defined for I/O automata (see, e.g., [Tut87]).

Definition 8.1.1. *Let $\mathcal{C} = (P, (\Gamma_{inp}, \Gamma_{out}, \Gamma_{int}), \gamma, J)$ be a component automaton and let Δ be an alphabet disjoint from P. Then*

*the Δ-*hiding version *of $\mathcal{C}$ is denoted by $\mathcal{C}_H^\Delta$ and is defined as $\mathcal{C}_H^\Delta = (P, (\Gamma_{inp} \setminus \Delta, \Gamma_{out} \setminus \Delta, \Gamma_{int} \cup \Delta), \gamma, J)$.* □

Composability is in general not preserved by the operation of hiding since composability requires the internal actions of the component automata to belong to one component automaton only, whereas external actions are not subject to such a restriction. The Δ-hiding version of a team automaton thus need not be a team automaton over the Δ-hiding versions of its original constituting component automata. For our composable system $\mathcal{S}$ and subsets $\Delta_i \subseteq \Sigma_{i,ext}$, for all $i \in \mathcal{I}$, the system $\mathcal{S}' = \{(\mathcal{C}_i)_H^{\Delta_i} \mid i \in \mathcal{I}\}$ is composable if and only if for all $i \in \mathcal{I}$, $\Delta_i \cap \bigcup_{j \in \mathcal{I} \setminus \{i\}} \Sigma_{j,ext} = \varnothing$.

The external actions that are to be hidden are those that are only used for communications between certain component automata and that should not be available for communication with other component automata.

Definition 8.1.2. *A pair $\mathcal{C}_i, \mathcal{C}_j$, with $i, j \in \mathcal{I}$, is communicating (in $\mathcal{S}$) if there exists an $a \in (\Sigma_{i,ext} \cup \Sigma_{j,ext})$ such that*

$$a \in (\Sigma_{i,inp} \cap \Sigma_{j,out}) \cup (\Sigma_{j,inp} \cap \Sigma_{i,out}).$$

Such an a is called a communicating action (in $\mathcal{S}$). By Σ_{com} we denote the set of all communicating actions (in $\mathcal{S}$). □

Note that the *communicating relation* between component automata, i.e. the set of all pairs of communicating component automata over component automata, is symmetric and irreflexive. Note furthermore that the fact that an action is communicating does not imply that a team automaton over $\mathcal{S}$ will actually have a synchronization involving this action as a communication, i.e. in its two roles of input and output. The communicating property is based solely on alphabets and is thus by no means related to transition relations.

With the hide operation we can internalize all communicating actions of a team automaton, before this team automaton is used to build a higher-level team automaton. The result is a team automaton that is closed with respect to its communications to the outside world.

Definition 8.1.3. *The* (communication) closed version *of $\mathcal{T}$ is denoted by $\boxed{\mathcal{T}}$ and is defined as*

$$\boxed{\mathcal{T}} = \mathcal{T}_H^{\Sigma_{com}}.$$ □

Rather than the team automaton itself we may now use its closed version in a new construction. If we do this, then only those output (input) actions that do not have a matching input (output) action within the team automaton are external actions of the closed version of the team automaton. The remaining external actions have been reclassified as internal actions.

In practice one often wants to work with several copies of a component automaton. In our model, however, more than one copy of a component automaton in a set of component automata in general means that this set does not satisfy composability. An operation renaming the actions of a component automaton solves this problem. Modulo renaming, these copies all have the same computations (and thus exhibit the same behavior). The operation of renaming has also been defined for I/O automata (see, e.g., [Tut87]).

Recall that a function $f : A \to A'$ is a bijection if it is injective ($f(a_1) \neq f(a_2)$ whenever $a_1 \neq a_2$) and surjective (for every $a' \in A'$ there exists an $a \in A$ such that $f(a) = a'$).

Definition 8.1.4. *Let $\mathcal{C} = (P, (\Gamma_{inp}, \Gamma_{out}, \Gamma_{int}), \gamma, J)$ be a component automaton, let Δ be an alphabet disjoint from P, and let $h : (\Gamma_{inp} \cup \Gamma_{out} \cup \Gamma_{int}) \to \Delta$ be a bijection. Then*

the h-renamed version *of $\mathcal{C}$ is denoted by $\mathcal{C}_N^h$ and is defined as $\mathcal{C}_N^h = (P, (h(\Gamma_{inp}), h(\Gamma_{out}), h(\Gamma_{int})), \{(q, h(a), q') \mid (q, a, q') \in \gamma\}, J)$.* □

In practice, an h-renamed version of a component automaton might best be defined to generate new names which are disjoint from the domain set, e.g. by requiring Δ to be disjoint from its alphabet.

It is clear that, apart from the use of new names, certain properties of team automata continue to hold for their h-renamed versions.

Lemma 8.1.5. *Let h be a bijection such that $\mathcal{T}_N^h$ is the h-renamed version of $\mathcal{T}$. Then*

(1) $\mathbf{C}^{\infty}_{\mathcal{T}_N^h} = \widehat{h}(\mathbf{C}^{\infty}_{\mathcal{T}})$, where $\widehat{h}$ is the extension of h to $\Sigma \cup Q$ defined by $\widehat{h}(q) = q$, for all $q \in Q$,

(2) $\mathbf{B}^{\Sigma,\infty}_{\mathcal{T}_N^h} = h(\mathbf{B}^{\Sigma,\infty}_{\mathcal{T}})$, and

(3) if an action a is free (ai, si, sipp, wipp, sopp, wopp, ms, sms, wms) in $\mathcal{T}$, then $h(a)$ is free (ai, si, sipp, wipp, sopp, wopp, ms, sms, wms) in $\mathcal{T}_N^h$. □

In the next subsection we show how to apply the operations introduced here.

8.1.2 GROVE Document Editor Architecture

In [Ell97] the distributed architecture of the GROVE document editor (see, e.g., [EGR90]) — depicted here in Figure 8.1 — is discussed. In this section we show how to model this architecture using a formal description in terms of team automata. In the process we point out where the notions introduced in the previous subsection come into play.

We are given a user interface automaton $\mathcal{C}_1$, a keeper automaton $\mathcal{C}_2$, an application automaton $\mathcal{C}_3$, and a coordination automaton $\mathcal{C}_4$. These together form a composable system $\mathcal{S} = \{\mathcal{C}_i \mid i \in [4]\}$. Only the pairs $\mathcal{C}_i, \mathcal{C}_{i+1}$, $i \in [3]$,

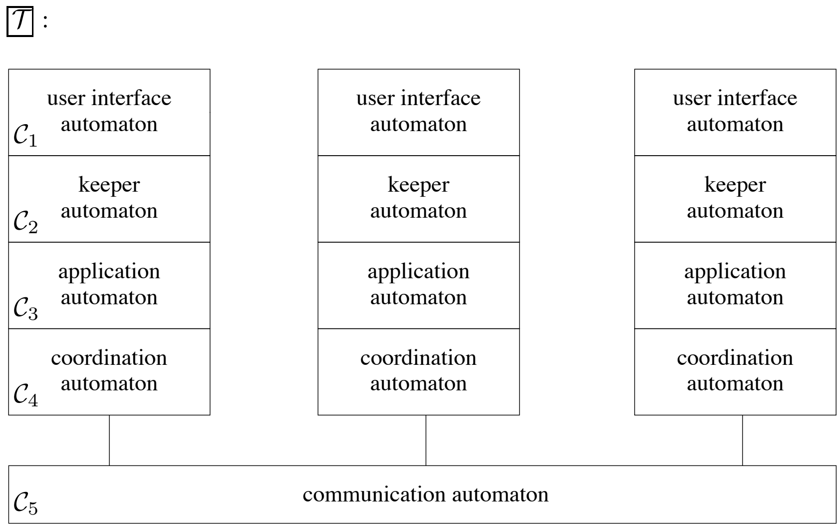


Fig. 8.1. The GROVE document editor architecture.

are communicating. All external actions of $\mathcal{C}_2$ and $\mathcal{C}_3$ are communicating in $\mathcal{S}$. $\mathcal{C}_1$ has external actions that are not communicating in $\mathcal{S}$, but intended to be used solely for interaction with the users. $\mathcal{C}_4$ has external actions to be used for communication with the communication automaton $\mathcal{C}_5$, which is to be added in a later stage. However, the non-communicating actions of $\mathcal{C}_1$ are different from those of $\mathcal{C}_4$.

The architecture requires all components in $\mathcal{S}$ to synchronize on all communications, thus we construct the *maximal-ai*-team automaton $\mathcal{T}$ over $\mathcal{S}$. Then this team automaton $\mathcal{T}$ is closed, resulting in its closed version $\boxed{\mathcal{T}}$. Now all communicating external actions are internal in $\boxed{\mathcal{T}}$. In this way we prohibit further synchronizations involving a component of $\mathcal{S}$. The only remaining external actions are those of $\mathcal{C}_1$ and those of $\mathcal{C}_4$.

Next we introduce several renamed versions of $\boxed{\mathcal{T}}$ satisfying the following two conditions.

First, the sets of actions of the renamed versions should be mutually disjoint in order to avoid undesired synchronizations of their user interfaces, and of actions to be used for the interaction with the communication automaton $\mathcal{C}_5$. Note that this condition ensures that these renamed versions form a composable system $\mathcal{S}'$.

Secondly, the external actions of $\boxed{\mathcal{T}}$ originating from the coordination automaton $\mathcal{C}_4$ should be renamed in such a way that they will communicate with actions from $\mathcal{C}_5$.

Finally, to obtain the desired team automaton modeling the GROVE document editor architecture we define a team automaton over $\mathcal{S}'' = \{\mathcal{C}_5\} \cup \mathcal{S}'$. Since we want $\mathcal{C}_5$ to communicate with all renamed versions of $\boxed{\mathcal{T}}$ we construct the *maximal-ai*-team automaton over $\mathcal{S}''$, which thus results in all communicating actions being synchronized.

It is clear that the iterated way in which we have constructed this final team automaton guarantees that no undesired synchronizations between, e.g., a keeper automaton and the communication automaton can take place. Not only all communication between the communication automaton and any of the renamed versions of $\boxed{\mathcal{T}}$ takes place via their coordination automata, but also there are no interactions between the renamed versions of $\mathcal{T}$. This is conveniently modeled by the communication closure. Moreover, the explicit construction used to form the final team automaton makes all communications mandatory.

8.1.3 Conclusion

In this section we have seen how team automata can be used to model both the conceptual and the architectural level of groupware systems. Actually, many of the concepts and techniques of computer science, such as concurrency control, user interfaces, and distributed databases, need to be rethought in the groupware domain. Team automata are thus helpful for this rethinking. The team automata framework allows one to separately specify the components of a groupware system and to describe their interactions. It is thus neither a message-passing model nor a shared-memory model, but a shared-action model. In particular, we have seen that team automata provide us with tools allowing formal and precise definitions of various basic groupware notions.

One way of viewing the team automaton framework is as having a two-way mechanism to model a spectrum of group interactions. On the one hand we have peer-to-peer types of synchronization, in which all participants are considered equal. They model the group collaboration aspect that frequently occurs in synchronous groupware. On the other hand there are master-slave types of synchronization, in which output as a master may force the concurrent execution of a corresponding input action. They can be used to model asynchronous cooperation, as in workflow systems to enact certain modules (see, e.g., [EN93]).

Team automata thus fit nicely with the needs and the philosophy of groupware and thanks to the formal setup, theorems and methodologies from automata theory can be applied.

8.2 Team-Based Model Development

Software configuration management is a subfield of *software engineering* that deals with organizing and controlling evolving software systems throughout their life cycle (see, e.g., [IEEE93]). Through software configuration management models, technical and administrative direction and surveillance over the life cycle of software systems is given in order to identify the functional and physical characteristics of modules and their assemblies, to control releases and changes, to record the product status, and to validate the completeness, consistency, and conformance to specifications of the product. Incorporated are also areas such as construction management, process management, and team work control (see, e.g., [Dar91]).

Since software systems are becoming more and more complex, it is inevitable to parallelize the development of models for these systems in such a way that several teams of developers must work in parallel on (parts of) the model under design. At some point in time the efforts of these teams however need to be integrated and this, more often than not, leads to conflicts. Obviously, these conflicts need to be resolved. However, most of the time they are difficult and time consuming to resolve and furthermore they often require manual modeler intervention.

8.2.1 A Conflict-Free Cooperation Strategy

Software configuration management models use a *cooperation strategy* to ensure that changes are coordinated such that one change does not — unwillingly — undo or conflict with the effects of another change. A *conservative cooperation strategy* prevents conflicting changes by using a simple locking scheme: developers working on a specific module version or configuration can lock it against further changes, and while a version or configuration is locked other developers are excluded from creating new versions. On the contrary, in an *optimistic cooperation strategy* each developer is active in his or her own workspace and various versions of the same module can be created.

Both conservative and optimistic cooperation strategies eventually need to merge parallel changes. Existing approaches of merges often lack early conflict detection, which results in conflicts becoming apparent only during the actual merges. These conflicts then have to be resolved, which is very time consuming. A conservative cooperation strategy does reduce the potential number of conflicts, since each part of a model may only be changed by one team *at a time* (the situation where two or more teams are working at cross purposes is avoided). However, a change to one part can affect all dependent parts and unfortunately thus still lead to conflicts during merge.

We note that problems during merge are avoided if we have a precise definition of when a change to a part is local, i.e. when the change only affects that part and not the rest of the model. When using an optimistic strategy, each part is edited in its own workspace by one unique team of developers. If we thus require each team to make local changes only to its own part, then integration becomes straightforward and, in fact, can be done automatically due to the absence of conflicts. We call this a *conflict-free (cooperation) strategy*.

We now illustrate our conflict-free strategy for the development of an object-oriented model. As parts of the model we use packages of classes, which are commonly used to structure a model (see, e.g., [RBP$^+$91] and [UML99]). A notion of local change can, e.g., be defined through invariancy of the services offered through the interface of the package. The interface is then the contract of the package with the rest of the model ([Mey92]).

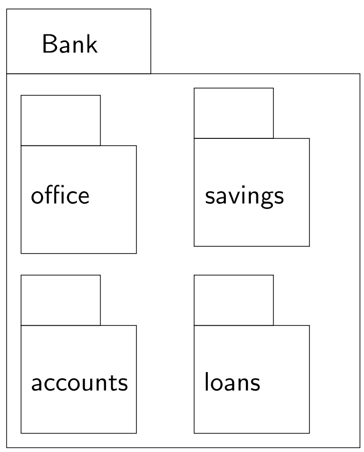


Fig. 8.2. The departments of a bank.

In Figure 8.2 we present part of a model in which a package Bank models a real-life bank (the figure is drawn using the notation of [UML99]). Four of its departments are modeled as subpackages. Bank can be developed in parallel by four teams where each team separately develops one of the departments. The changes made to each department are local and the merge to form the modified bank is straightforward. Note that these packages can be developed in entirely different geographic locations. Each team has its own workspace to make its changes and is only dependent on the other teams during merge.

We use an optimistic strategy, but we constrain the changes in each workspace to prevent conflicts during merge. A model is split into several views for individual development and later merge. In this case we however block changes to the views which cause conflict during merge. In any realistic project, however, the connections between the parts (packages) of the model cannot stay the same during the complete life cycle of the model. Modifications requiring non-local changes of packages (thus invalidating the conflict-free strategy) need to occur and hence a conflict-free merge cannot be guaranteed. These changes can however be localized by (temporarily) adding a new package, which contains those original packages between which changes have to be made. These changes are then local with respect to the newly added package and thus allow for the conflict-free strategy to be applied to the model with the extra package. This is illustrated in Figure 8.3.

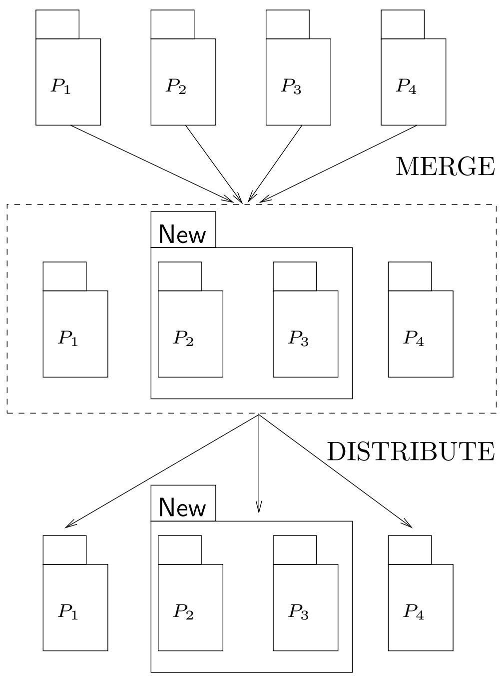


Fig. 8.3. A package is added.

The packages P_1 to P_4 are edited using the conflict-free strategy. However, non-local changes are required between packages P_2 and P_3. The work under development is merged and a temporary package New is added to group these two wayward parts. Note that because up to now the changes to the packages of the model have been local, the merge is without conflicts. The model is consequently redistributed with the new structure and work can continue under the conflict-free strategy, since changes are once again local. The extra package can be removed once the new connections between the wayward packages are stable.

Note that in practice it may not be necessary to merge all the packages under development. It may be sufficient only to merge the packages for which the non-local changes are required to form a partial model, e.g., if each package is at most once the subject of such a temporary merge before a complete intermediary model is produced.

The architecture of a model thus is initially determined by top-down decomposition. This architecture can however be adapted to suit the need of our strategy. We call this part of the conflict-free strategy the *renegotiation phase*. Too many of such phases during the model's life cycle are inconvenient. They however indicate that the high-level architecture of the model is not yet stable, or even that the model is as yet too premature to be developed in a distributed fashion. Ideally, the initial breakdown of the model into packages should only be done by experienced modelers, thereby reducing the number of renegotiation phases as much as possible. The initial model should consequently be developed in one workspace until there is enough confidence that a right choice has been made for a stable enough architecture, after which the conflict-free strategy can be applied to it. The same considerations hold when one of the packages used in the conflic-free strategy is further split up into two or more subpackages for further parallel development.

8.2.2 Teams in the Conflict-Free Strategy

The decomposition of a model into packages is also used to dictate the structure of the team of developers working on the model. Each such team works on a distinct package of the model, i.e. for n packages we will have n teams working in parallel under the conflict-free strategy, each on one of these distinct packages. Packages can be hierarchical, i.e. a package can contain other packages. We have seen an example of this in Figure 8.2. We use this hierarchical structuring of a package to likewise structure the teams working on the model under the conflict-free strategy. Teams, in our approach, can be hierarchical and the hierarchical decomposition of a package naturally leads to the decomposition of the team working on the package into subteams.

Consider the hierarchical package P as sketched in Figure 8.4. It contains the subpackages $P_{1,1}$ and $P_{1,2}$ and each of these subpackages is further split up into two smaller subpackages ($P_{2,1}$ and $P_{2,2}$, and $P_{2,3}$ and $P_{2,4}$, respectively). A team T is working (exclusively) on package P, as indicated by the dotted arrow from P to T. This team T is split up into two teams that work on the two subpackages of P, and one of these teams is further split up, as dictated by the package architecture. The conflict-free strategy is thus used to manage the efforts of T together with the other teams working on the other packages. The same strategy is also used within the hierarchical package P to internally structure the efforts of team T using subteams. Note that this is not required: we have not further split up team $T_{1,2}$ because we have chosen to keep one large team to work on the entire package $P_{1,2}$. The conflict-free strategy can thus be used to parallelize the development of the model into parts, up to the number of packages that exist in a model at the deepest level of nesting. The choice of packages then partially dictates the structure of the teams.

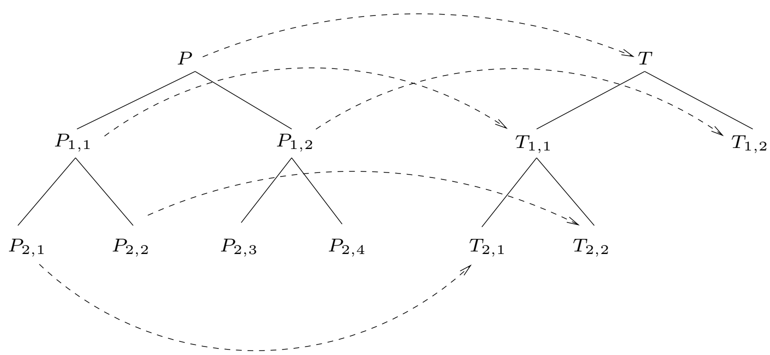


Fig. 8.4. Hierarchical teams.

Note that during a renegotiation phase the team structure is affected to reflect the new distribution of packages. In Figure 8.3, we (temporarily) merge the teams working on packages P_2 and P_3 in order to reflect the fact that they are now working together to determine the new interactions between these packages. Hence, the initial team structure is determined by the architecture of the initial model and is adapted dynamically due to renegotiation. In the example of Figure 8.3, the wayward packages P_2 and P_3, which are edited by the teams T_2 and T_3, respectively, are temporarily placed in a package New during renegotiation. These two teams together are then responsible for modifying this new package, as sketched in Figure 8.5.

The structure of the model and the structure of the teams are thus tightly coupled. The initial model determines how the teams can be distributed over the packages for parallel development. On the other hand, desired non-local changes of one of the teams can lead to a (temporary) change in architecture. The model itself is "actively" involved in the development process. This

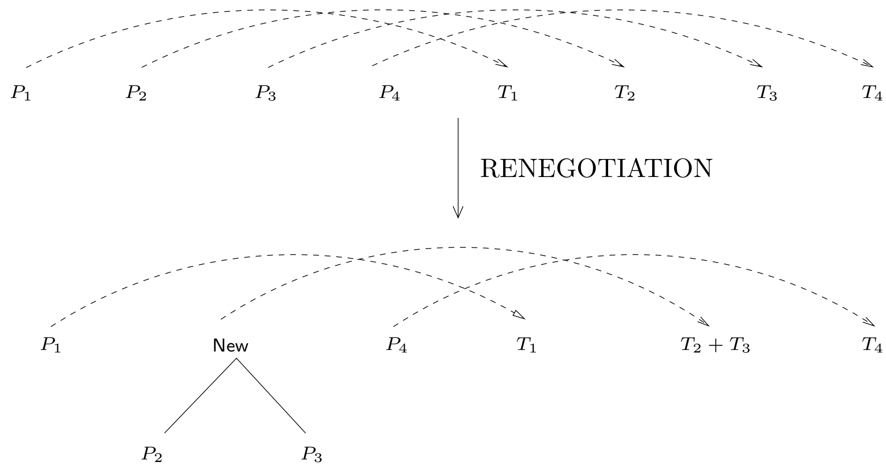


Fig. 8.5. Merging teams.

contrasts with many workflow or software process models, where the model under development is not really relevant (see, e.g., [KB95] and [DKW99]). They focus more on the documents to be produced, and their timing. The contents of these documents however do not play an explicit role.

In our approach, the activities of the teams can be divided into two categories: those which are internal to a team and those which involve other teams (due to renegotiation). The management of the teams in the conflict-free strategy can be divided along these lines. On the one hand, management can be localized and is only concerned with coordinating the changes to one package by one team. Here the focus is on coordinating a relatively small group in a well-defined context. On the other hand, the structure of the teams can be a separate management concern. The management of the hierarchical structure of the model and of the teams as given in Figure 8.4 can become an issue in its own right. This is a relatively more complex job than "just" managing one team. Seniority and experience can come into play when determining which role is played by which individual. Relatively unexperienced individuals should manage relatively small teams such as $T_{2,1}$, while a more experienced manager should lead the more complex team $T_{1,1}$. The most experienced manager can decide whether changes leading to renegotiation fit within the direction the model should be heading in order to match its specification.

Note that we do not discuss how teams should be led. We postulate a group of people who together perform a common editing of one package. We do not claim that they should coordinate their work in any specific way.

We just define the extent of their possible changes by only allowing local changes. We also do not discuss how two separate teams, when integrated, should coordinate their efforts. This is a nontrivial task, especially if the two teams previously worked according to different philosophies. We just constrain the extent of their possible actions as a new, larger team. This is a topic of research with strong sociological impact, which is however outside the scope of this thesis, but naturally fits well within CSCW. The conflict-free strategy does provide a context within which knowledge about how people work can be embedded.

8.2.3 Teams Modeled by Team Automata

We now sketch how a hierarchical team structure, as induced by the structure of the model under development in the way described in the previous subsections, can be modeled in terms of team automata. We interpret actions as operations or changes of (a package of) the model. Since internal actions of a component automaton cannot be observed by any other component automaton, these actions are ideally suited for representing a local change to a package using the conflict-free strategy. The external actions, on the other hand, are ideal for modeling the collaboration between packages.

In Figure 8.6 we represent our example teams T_2 and T_3 by two quite trivial component automata $\mathcal{T}_2$ and $\mathcal{T}_3$, respectively. The states of $\mathcal{T}_2$ are p_1, p_2, and p_3, whereas q_1 and q_2 are the states of $\mathcal{T}_3$. The wavy arcs indicate the initial states p_1 and q_1 of $\mathcal{T}_2$ and $\mathcal{T}_3$, respectively. $\mathcal{T}_2$ has no input actions, output actions a and d, and internal actions b and c, while $\mathcal{T}_3$ only has output actions, viz. a and d. Their transition relations are as depicted in Figure 8.6. Now a possible scenario could be as follows. First $\mathcal{T}_2$ and $\mathcal{T}_3$ execute output action a in parallel. Consequently $\mathcal{T}_2$ executes a number of internal actions (i.e. local changes to its package without consulting the other teams). Eventually both component automata can execute output action d in parallel, after which this procedure can be repeated. Naturally we could imagine also $\mathcal{T}_3$ having some internal actions (i.e. local changes) to execute once in a while.

Note that $\{\mathcal{T}_2, \mathcal{T}_3\}$ is a composable system. In Figure 8.7, the state-reduced version $(\mathcal{T}_{2,3})_S$ of a team automaton $\mathcal{T}_{2,3}$ over $\{\mathcal{T}_2, \mathcal{T}_3\}$ is given. Note that output actions a and d are *sopp* in $\mathcal{T}_{2,3}$, requiring both $\mathcal{T}_2$ and $\mathcal{T}_3$ to change state, whereas only $\mathcal{T}_2$ is changing state when internal actions b or c are executed. The behavior of both $\mathcal{T}_2$ and $\mathcal{T}_3$ is thus reflected in the behavior of $\mathcal{T}_{2,3}$. In our interpretation, such peer-to-peer types of synchronization can represent changes which affect two or more packages, i.e. non-local changes. The external actions of $\mathcal{T}_{2,3}$ thus represent the shared operations on the

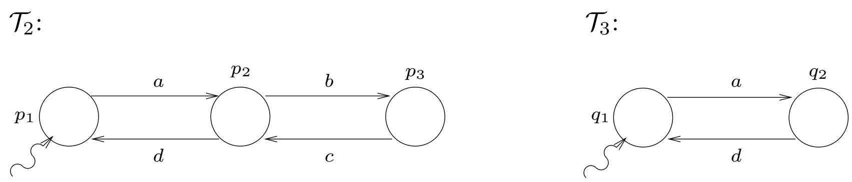


Fig. 8.6. Component automata $\mathcal{T}_2$ and $\mathcal{T}_3$.

merged packages P_2 and P_3. Note that we could also use master-slave types of synchronization to model boss-employee relations in which employees have to follow orders from their bosses.

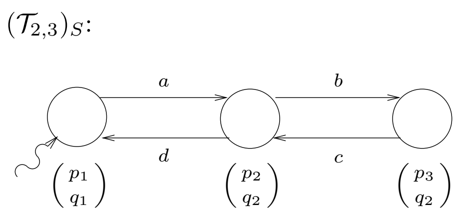


Fig. 8.7. State-reduced team automaton $(\mathcal{T}_{2,3})_S$ over $\{\mathcal{T}_2, \mathcal{T}_3\}$.

The external actions of $\mathcal{T}_{2,3}$ consequently can be hidden in order to obtain a team automaton with only internal actions, i.e. with only local operations on its packages. The resulting team automaton can then be used as a component automaton in a larger team automaton. In this way, subteams and hierarchical team structures can be modeled. In Figure 8.8, e.g., team automaton $\mathcal{T}$ is defined as a composition of team automaton $\mathcal{T}_{2,3}$ with certain component automata $\mathcal{T}_1$ and $\mathcal{T}_4$. As such, team automata are well suited for modeling (the actions of) the hierarchical teams in the conflict-free strategy.

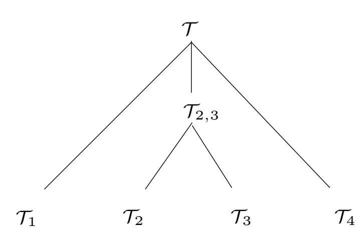


Fig. 8.8. A team automaton $\mathcal{T}$ over $\mathcal{T}_1$, $\mathcal{T}_{2,3}$, and $\mathcal{T}_4$.

8.2.4 Conclusion

In this section we have discussed a conflict-free strategy for the development of a model by several teams of developers working in parallel on distinct packages of the model. We guided the changes made by each team so as to ensure no conflicts occur during the merge of the produced efforts. This approach is scalable as each package can be developed in a similar fashion by splitting the package up further. We have moreover shown how packages under development can (temporarily) be merged during a renegotiation phase, if we need changes to a package that would invalidate the conflict-free strategy.

Additionally, we have discussed how the hierarchical structure of the model in packages can be used to structure the teams working on the model. The top-down decomposition of a model into packages guides the decomposition of the people working on the model into similarly structured teams. The renegotiation phase, when packages are temporarily merged, then gives heuristics on how the teams should further cooperate to implement changes without generating conflicts. We have sketched how this can formally be modeled by team automata.

The conflict-free strategy, along with the explicit discussion on the team structure and its actions, brings the worlds of CSCW, software engineering, software configuration management, and process modeling very close together. We have discussed how a large model can be developed and how the work between the people doing the actual work can be coordinated. Special to the approach is that the subject of the work, the model under development, is used to structure the work and thus plays an active part in deciding which changes are possible.

8.3 Spatial Access Control

As the complexity of reactive (computer) systems continues to increase, abstractions tend to be especially useful. For this reason, computer science often introduces and studies various models of computation that allow enhanced understanding and analysis. Computer science has also created a number of interesting metaphors (e.g., the desktop metaphor) that aid in end user understanding of computing phenomena. This section is concerned with a model and a metaphor. The model is team automata and the metaphor is *spatial access control*, which is based upon current notions of virtual reality, and helps demystify concepts of access control matrices and capability structures for the end user ([BB99]).

Our aim here is to connect the metaphor of spatial access control to the framework of team automata, and to show through examples how this combination facilitates the identification and unambiguous description of some key issues of access control. The rigorous setup of the framework of team automata allows one to formulate, verify, and analyze general and specific logical properties of various control mechanisms in a mathematically precise way. In realistically large (computer) systems, security is a big issue, and team automata allow formal proofs of correctness of its design. Moreover, a formal approach as provided by the team automata framework forces one to unambiguously describe control policies and it may suggest new approaches not seen otherwise. There is a large body of literature concerning topics like security, protection, and awareness in (computer) systems. Although team automata are potentially applicable also to these areas, we are currently not concerned with issues outside of spatial access control. We will conclude with a discussion of some variations and extensions of our setup.

We now begin by discussing the spatial access control metaphor by means of an example and subsequently we show how certain spatial access control mechanisms can be made precise and given a formal description using team automata. We first introduce information access modeling by granting and revoking access rights, and show how *immediate* versus *delayed revocation* can be formulated. Subsequently we extend our study to the more complex issue of meta access control and, finally, we show how team automata can deal with *deep* versus *shallow revocation*.

8.3.1 Access Control

A vital component of any (computer) system or environment is security and information *access control*, but this is sometimes done in a rather ad hoc or inadequate fashion with no underlying rigorous, formal model. In typical electronic file systems, access rights such as read-access and write-access are allocated to users on some basis such as “need to know”, ownership, or ad hoc lists of accessors. Within groupware systems, there are typically needs for more refined access rights, such as the right to scroll a document that is being synchronously edited by a group in real time. Furthermore, the granularity of access must sometimes be more fine grained and flexible, as within a software development team. Moreover, it is important to control access meta rights. For example, it may be useful for an author to grant another team member the right to grant document access to other non-team members (i.e. delegation). Various models have been proposed to meet such requirements (see, e.g., [SD92], [Rod96], and [Sik97]).

We use a spatial access metaphor based upon work of Bullock and colleagues in [BB97] and [BB99]. There, access control is governed by the rooms, or spaces, in which subjects and objects reside, and the ability of a subject to traverse space in order to get close to an object. Bullock also implemented a system called SPACE to test out some of these ideas ([Bul98]). A basic tenet of the SPACE access model is that a fundamental component of any collaborative environment is the environment itself (i.e. the space). It is the shared territory within which information is accessed and interaction takes place. Often this shared space is divided into numerous regions that segment the space. This allows decomposition of a very large space into smaller ones for manageability. It also allows cognitive differentiation (i.e. different concerns, memories, and thoughts associated with different regions), and distributed implementation (i.e. different servers for different regions).

By adopting a spatial approach to access control, the SPACE metaphor exploits a natural part of the environment, making it possible to hide explicit technical security mechanisms from end users through the natural spatial makeup of the environment. These users can then make use of their knowledge of the environment to understand the implicit security policies. Users can thus avoid understanding technical concepts such as so-called access matrices, which helps to avoid misunderstandings.

We consider here a virtual reality, in which a user can traverse from room to room by using keyboard keys, the mouse, or fancier devices. It is a natural and simple extension to assume that access control checking happens at the boundaries (doors) between spaces (rooms) when a user attempts to move from one room to another. If the access is OK, then the user can enter and use the resources associated with the newly entered room.

To illustrate the various concepts throughout this section, we present a simple running example which is concerned with read and write access to a file F by a user Kwaku. This file might be any data or document that is stored electronically within a typical file system. The file system keeps track of which users have which access rights to the file F. Three types of access rights are possible for a file F: null access (implying the user can neither read nor write the file), read access (implying the user cannot write the file), and full access (implying the user can read and write — i.e. edit — the file).

In security literature, authentication deals with verification that the user is truly the person represented, whereas authorization deals with validation that the user has access to the given resource. Assume that when Kwaku logs into the system, there is an authentication check. Then whenever he tries to read or write F, authorization checking occurs, and Kwaku is either allowed the access, or not. Using the SPACE metaphor, the above three types of access rights can be associated with three rooms as shown in Figure 8.9.

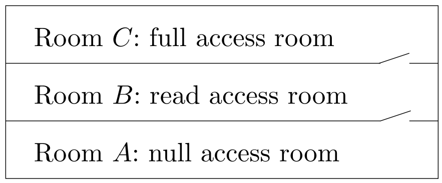


Fig. 8.9. A rooms metaphor for access control.

Room A is associated with no access to the document, room B is associated with read access, and room C models full access. Suppose Kwaku is in room B, the reading room. Presence in this room means that any time Kwaku decides to read F, he can do so. However, if he attempts to make changes to F, then he will fail because he does not have write access in room B. There are doors between rooms, implying that user access rights can be dynamically changed by changing rooms. We discuss this dynamic change in more detail later in this section.

This access mechanism satisfies a number of end user friendly properties: it is simple, understandable by non-computer people, relatively natural and unobtrusive, and elegant. Later we show how modeling this type of access metaphor via team automata adds precision, mathematical rigor, and analytic capabilities.

We now show how to model our access control example in the team automata framework. The component automaton $\mathcal{C}^C$ depicted in Figure 8.10(a) corresponds to room C of Figure 8.9, as it models full access to file F. The states of $\mathcal{C}^C$ are C_e modeling an empty room, C_n modeling F is not accessed, C_r modeling F is being read, and C_w modeling F is being written (edited). The wavy arc in Figure 8.10(a) denotes the initial state C_e. The actions of $\mathcal{C}^C$ are e_{BC} (enter room), e_{CB} (exit room), r^C (begin reading), $\underline{r}^C$ (end reading), w^C (begin writing), and $\underline{w}^C$ (end writing).

$\mathcal{C}^C$ thus has the transitions (C_e, e_{BC}, C_n), (C_n, e_{CB}, C_e), (C_n, r^C, C_r), $(C_r, \underline{r}^C, C_n)$, (C_r, w^C, C_w), and $(C_w, \underline{w}^C, C_r)$. Now transition (C_e, e_{BC}, C_n), e.g., shows that in $\mathcal{C}^C$ we can go from state C_e to C_n by executing action e_{BC}. We also see that transitioning directly from C_n to C_w is not possible. Furthermore, entering and exiting room C may only occur via state C_n. We choose to specify actions r^C, $\underline{r}^C$, w^C, and $\underline{w}^C$ as internal actions of $\mathcal{C}^C$, and e_{BC} and e_{CB} as external actions of $\mathcal{C}^C$. Both e_{BC} and e_{CB} clearly should be externally visible and therefore cannot be internal. For the moment we

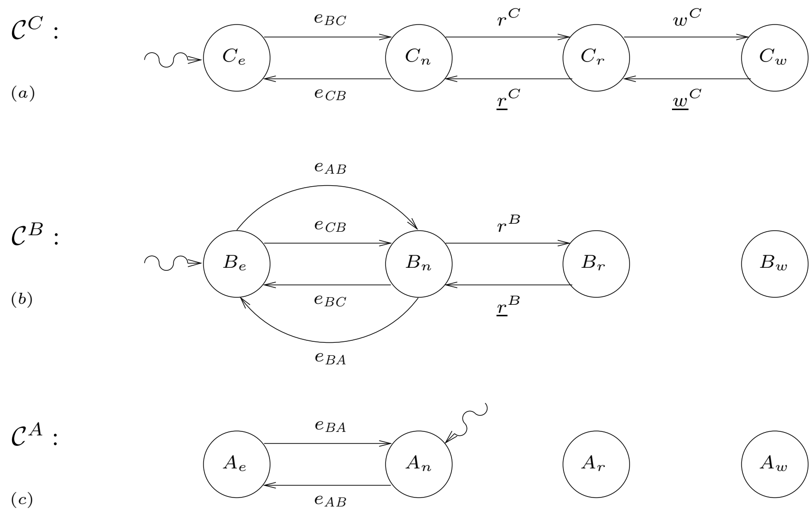


Fig. 8.10. Component automata $\mathcal{C}^C$, $\mathcal{C}^B$, and $\mathcal{C}^A$: rooms C, B, and A.

choose them to be output actions. These two external actions are candidates for being synchronized with actions of the same name in other component automata when forming a team automaton over $\mathcal{C}^C$ and the two component automata described next.

Component automata $\mathcal{C}^B$ and $\mathcal{C}^A$ corresponding to rooms B and A, respectively, are somewhat similar to $\mathcal{C}^C$. However, write access is denied in rooms B and A and read access is denied in room A. Component automata $\mathcal{C}^B$ and $\mathcal{C}^A$ are depicted in Figure 8.10(b,c). Note that $\mathcal{C}^A$ has initial state A_n (hence initially room A is not empty) and that both $\mathcal{C}^B$ and $\mathcal{C}^A$ have states unreachable from the initial state. Actions r^B and $\underline{r}^B$ are internal, while the rest of the actions of $\mathcal{C}^B$ and $\mathcal{C}^A$ are external (output) actions.

Now we want to combine $\mathcal{C}^C$, $\mathcal{C}^B$, and $\mathcal{C}^A$ into one team automaton reflecting a given access policy. They clearly form a composable system $\{\mathcal{C}^C, \mathcal{C}^B, \mathcal{C}^A\}$ and we combine them into a team automaton $\mathcal{T}^{CBA}$ as follows. Since each state of $\mathcal{T}^{CBA}$ is a combination of a state from $\mathcal{C}^C$, a state from $\mathcal{C}^B$, and a state from $\mathcal{C}^A$, $\mathcal{T}^{CBA}$ has $4^3 = 64$ states. Initially $\mathcal{T}^{CBA}$ is in state (A_n, B_e, C_e), which means one starts in room A, while rooms B and C are empty.

Assuming that one can have only one kind of access rights at a time, two of the rooms should be empty at any moment in time. This means that $\mathcal{T}^{CBA}$ should be defined in such a way that in each of its reachable states two of the three component automata are always in state "empty". We let the component automata synchronize on the external actions e_{AB}, e_{BA}, e_{BC}, and e_{CB}. Each such synchronized external action of $\mathcal{T}^{CBA}$ corresponds to exiting a room while entering another. Synchronization of action e_{AB}, e.g., models a move from room A to room B. This move is represented by the transition $((A_n, B_e, C_e), e_{AB}, (A_e, B_n, C_e))$ showing that in component automaton $\mathcal{C}^A$ we exit room A, in automaton $\mathcal{C}^B$ we enter room B, and in component automaton $\mathcal{C}^C$ we do nothing (i.e. remain idle). This represents a change in access rights from null access (in room A) to read access (in room B). We do not include, e.g., the transition $((A_n, B_e, C_e), e_{AB}, (A_e, B_e, C_e))$ which would let the user exit room A but never enter room B. Furthermore, the user could be in more than one room at a time if we would allow transitions like $((A_n, B_e, C_e), e_{AB}, (A_n, B_n, C_e))$. In T^{CBA} we include only the four transitions representing the synchronized changing of rooms. In each of these transitions, one component automaton is idle. Since all internal (read and write related) actions are maintained, in each of these only that component automaton is involved to which such an action belongs.

The state-reduced version $\mathcal{T}_S^{CBA}$ of the thus defined team automaton $\mathcal{T}^{CBA}$ over $\{\mathcal{C}^C, \mathcal{C}^B, \mathcal{C}^A\}$ is depicted in Figure 8.11.

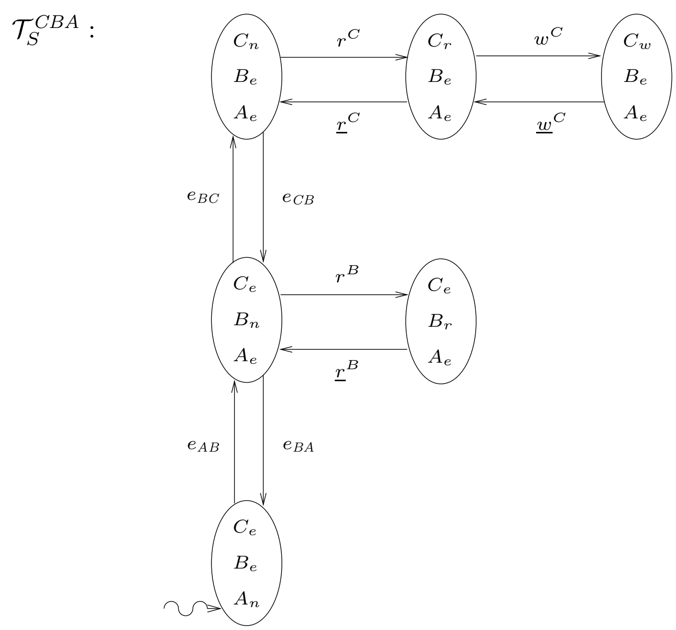


Fig. 8.11. State-reduced team automaton $\mathcal{T}_S^{CBA}$ over $\{\mathcal{C}^C, \mathcal{C}^B, \mathcal{C}^A\}$.

Recall that $\mathcal{T}^{CBA}$ is not the only team automaton over $\{\mathcal{C}^C, \mathcal{C}^B, \mathcal{C}^A\}$. Also recall that the decision to consider e_{AB}, e_{BA}, e_{BC}, and e_{CB} as output actions in all component automata of $\mathcal{T}^{CBA}$ was made more or less arbitrarily. In fact, it depends on how one views the action of entering and exiting a room within the team automaton $\mathcal{T}^{CBA}$. By choosing all of those actions to be output (and thus of the same type), exiting one room and entering another is seen as a *sopp* action. Recall that, on the other hand, master-slave types of synchronization occur when input actions can only occur as a response (slave) to output actions. In our example, assume that one views the changing of rooms as an action initiated by leaving a room and forcing the room that is entered to accept the entrance. Then one would name, e.g., e_{AB} an output action of $\mathcal{C}^A$ and an input action of $\mathcal{C}^B$, and e_{BA} an output action of $\mathcal{C}^B$ and an input action of $\mathcal{C}^A$. This causes both e_{AB} (with master $\mathcal{C}^A$ and slave $\mathcal{C}^B$) and e_{BA} (with master $\mathcal{C}^B$ and slave $\mathcal{C}^A$) to be *sms*. Likewise for the other actions.

In addition, Section 5.4 defines strategies that lead specifically to uniquely defined combinations of peer-to-peer and master-slave types of synchronization within team automata. The team automata framework allows one to model many other features useful in virtual reality environments. A door, e.g., can be extended to join more than two rooms since any number of component automata can participate in an output action. Furthermore, as said before, a user could be in more than one room at a time.

8.3.2 Authorization and Revocation

We continue our running example by adding Kwaku, a user whose access rights to file F will be checked by the access control system $\mathcal{T}^{CBA}$. Kwaku is represented by component automaton $\mathcal{C}^U$, depicted in Figure 8.12. This extension complicates our example in the sense that Kwaku's read and write access rights can be changed independently of his whereabouts. Only to enter a room he has to be authorized. Thus access rights are no longer equivalent with being in a room, but rather with the possibility to enter a room. To add this to the team automaton formalization, we will use the feature of iteratively constructing team automata with team automata as their constituting component automata.

Kwaku starts in state U_n with no access rights. The actions $m(r)$, $\underline{m}(r)$, $m(w)$, and $\underline{m}(w)$ model the (meta) operations of "being granted read access", "being revoked read access", "being granted write access", and "being revoked write access", respectively. Since these clearly are passive actions from Kwaku's point of view, we choose all of them to be input actions. Note that Kwaku can end up in state U_w if and only if he was granted access rights

$\mathcal{C}^U$:

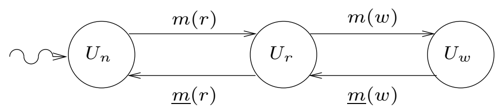


Fig. 8.12. Component automaton $\mathcal{C}^U$: user Kwaku.

to read and to write, i.e. actions $m(r)$ and $m(w)$ have taken place. When Kwaku's write access is consequently revoked by transition $(U_w, \underline{m}(w), U_r)$, he ends up in state U_r.

Now suppose that we want to model Kwaku's options for editing file F, which is protected by the access control system $\mathcal{T}^{CBA}$. Then we would like to compose a team automaton over $\mathcal{T}^{CBA}$ and $\mathcal{C}^U$. To do so, first note that $\{\mathcal{T}^{CBA}, \mathcal{C}^U\}$ is a composable system. Next we choose a transition relation, i.e. for each action a subset from its complete transition space in $\{\mathcal{T}^{CBA}, \mathcal{C}^U\}$ is selected, thereby formally fixing an access control policy for Kwaku under the constraints imposed by $\mathcal{T}^{CBA}$.

The initial state of any team over $\{\mathcal{T}^{CBA}, \mathcal{C}^U\}$ is (A_n, B_e, C_e, U_n), i.e. Kwaku is not yet editing F and is in the virtual room A without access rights. Now imagine the access rights to be keys. Hence Kwaku needs the right key to enter reading room B, i.e. action $m(r)$ must take place before action e_{AB} becomes enabled. This action $m(r)$ leads us from the initial state to (A_n, B_e, C_e, U_r). Now Kwaku has the key to enter room B by $((A_n, B_e, C_e, U_r), e_{AB}, (A_e, B_n, C_e, U_r))$. This transition models the acceptance of Kwaku's entrance of room B, i.e. this action is the authorization activity mentioned earlier. Hence our choice of the transition relation fixes the way we deal with authorization. If we would include, e.g., $((A_n, B_e, C_e, U_n), e_{AB}, (A_e, B_n, C_e, U_n))$ in the transition relation, this would mean that Kwaku can enter room B without having read access rights for F. Note however that since transitions involving internal actions of either $\mathcal{T}^{CBA}$ or $\mathcal{C}^U$ by definition cannot be pre-empted in any team over $\{\mathcal{T}^{CBA}, \mathcal{C}^U\}$, our transition relation must contain $((A_e, B_n, C_e, U_n), r^B, (A_e, B_r, C_e, U_n))$. Hence Kwaku, once in room B, can always begin reading file F. By not including $((A_n, B_e, C_e, U_n), e_{AB}, (A_e, B_n, C_e, U_n))$ in our transition relation we avoid that Kwaku can read F without ever having been granted read access. This leads to the question of the revocation of access rights.

As argued, (A_e, B_n, C_e, U_r) — meaning that Kwaku is in room B with reading rights — will be a reachable state. Now imagine that while in this state Kwaku's reading rights are revoked by $\underline{m}(r)$. To which state should this action lead, i.e. in what way do we handle revocation of access rights? We could opt for modeling *immediate revocation* or *delayed revocation*. The latter is what we have chosen to model first. Thus our answer to the question above is to include $((A_e, B_n, C_e, U_r), \underline{m}(r), (A_e, B_n, C_e, U_n))$. The result is that Kwaku can pursue his activities in room B, but cannot re-enter the room once he has left it (unless his read access has been restored). He is thus still able to read (browse) F, but the moment he decides to re-open the file this fails. Likewise, if Kwaku is writing F when his writing right is revoked, then he can continue editing (typing in) F, but he cannot re-enter room C as long as his write access right has not been restored. On this side of the revocation spectrum, a user can thus continue his or her current activity even when his or her rights have been revoked. He or she can do so until he or she wants to restart this activity, at which moment an authorization check is done to decide if he or she has the right to restart this activity. In some applications, this may be an intolerable delay.

Immediate revocation, on the other hand, means the following. If a user is reading when his or her reading right is revoked, then the file immediately disappears from view, while if a user is writing when his or her writing right is revoked, then the edit is interrupted and writing is terminated in the middle of the current activity. In some applications, this is overly disruptive and unfriendly. If we would want to incorporate immediate revocation into our example we would have to adapt our distribution of actions a bit. As said before, since r^B is an internal action we cannot disallow action r^B to take place after $((A_e, B_n, C_e, U_r), \underline{m}(r), (A_e, B_n, C_e, U_n))$ has revoked Kwaku's reading rights. If we instead choose r^B to be an external action, we are given the freedom not to include $((A_e, B_n, C_e, U_n), r^B, (A_e, B_r, C_e, U_n))$ in our transition relation. The result is that as long as Kwaku is not being granted read access by action $m(r)$, the only way left to proceed for Kwaku in state (A_e, B_n, C_e, U_n) is to exit room B by $((A_e, B_n, C_e, U_n), e_{BA}, (A_n, B_e, C_e, U_n))$. Modeling immediate revocation thus requires that actions such as r^B are visible, since in that way we can choose them not to be enabled in certain states. Immediate revocation also implies that we still want Kwaku to be able to stop reading and leave state (A_e, B_r, C_e, U_n) by $((A_e, B_r, C_e, U_n), \underline{r}^B, (A_e, B_n, C_e, U_n))$. Action $\underline{r}^B$ can thus remain internal.

This finishes the description of a part of a team automaton $\mathcal{T}$ over $\{\mathcal{T}^{CBA}, \mathcal{C}^U\}$. In Figure 8.13 the state-reduced version $\mathcal{T}_S$ of $\mathcal{T}$ (for delayed revocation) is depicted.

Recall that team automata are intended to be used to model (logical) design issues. An action can take place provided certain preconditions hold, and affects only states of those component automata involved in that action. Hence at this level there is no notion of time and no means are provided to give one action priority over another. A result of the lack of a notion of time

$\mathcal{T}_S$:

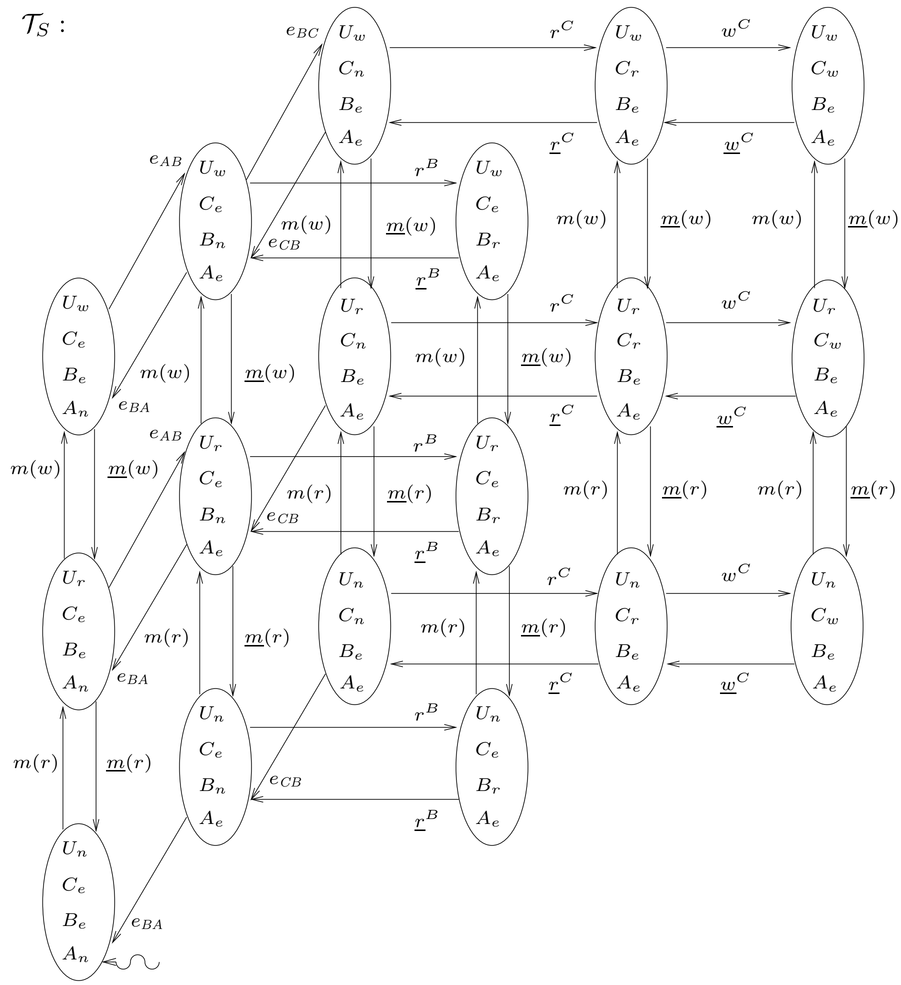


Fig. 8.13. Team automaton $\mathcal{T}_S$ over $\{\mathcal{T}^{CBA}, \mathcal{C}^U\}$.

is, e.g., that nothing can be said about how long it takes before Kwaku has left reading room B after his reading access right has been revoked. However, time and priorities may be added to the basic model as extra features.

Again, $\mathcal{T}$ is not the unique team automaton over $\{\mathcal{T}^{CBA}, \mathcal{C}^U\}$, but it is a team automaton one obtains by choosing a specific transition relation with a specific protocol in mind. Once again this shows that the freedom of the team automata model to choose transition relations offers the flexibility to distinguish even the smallest nuances in the meaning of one's design. Another interesting feature of the team automata framework is shown by the following application of the results proven in Section 5.2 to our running example. In whatever order one chooses to construct a team automaton over the component automata $\mathcal{C}^C$, $\mathcal{C}^B$, $\mathcal{C}^A$, and $\mathcal{C}^U$, we know that it will always be possible to construct the team $\mathcal{T}$ discussed above. This means that instead of first constructing $\mathcal{T}^{CBA}$ over $\{\mathcal{C}^C, \mathcal{C}^B, \mathcal{C}^A\}$, and then adding $\mathcal{C}^U$, we could just as well have constructed an iterated team by, e.g., starting from the user component automaton $\mathcal{C}^U$ and adding successively the component automata $\mathcal{C}^C$, $\mathcal{C}^B$, and $\mathcal{C}^A$ modeling the access rights that can be exercised. Moreover, independent of the way a team automaton over $\mathcal{C}^C$, $\mathcal{C}^B$, $\mathcal{C}^A$, and $\mathcal{C}^U$ is constructed, more component automata can be added.

As an example, suppose that Kwaku has other interests than the file F. Hence imagine a component automaton $\mathcal{T}^{NBA}$ in which he can transition into a state in which he plays some basketball. Then we may construct a team over the team automaton $\mathcal{T}$ just described and the component automaton $\mathcal{T}^{NBA}$ modeling when Kwaku is entitled — or perhaps even forced — to have a break (which is of some importance in these times of RSI). In general, new component automata can be added to a given team automaton at any moment of time, without affecting the possibilities of any new additions. We thus conclude once again that the team automata framework scores high on scalability. We will come back to this shortly.

8.3.3 Meta Access Control

Until now we have seen how team automata can be used to describe the control of a user's access to a file depending on his or her rights. Here we further elaborate on the granting and revoking of access rights and we consider *meta access control*. This means that privileges such as granting and revoking of rights can themselves be granted and revoked. The complicated (recursive) situations that may arise in this fashion depend on the chosen (meta) access control policy and we demonstrate how they can unambiguously and concisely be defined in terms of team automata.

Figure 8.14 shows a component automaton $\mathcal{C}^0$ that models a building with three levels — A, B, and C — corresponding to null access, read access, and full access, respectively. This component automaton shows the same access structure as the three rooms of Figure 8.10. Now, however, the status of the user directly determines the level he or she operates on and the granting and revoking of access rights is identified with changing levels. This differs from the previous example where the status of the user only determined his or her rights to enter a room.

Consequently, in $\mathcal{C}^0$ the user moves in two dimensions: vertically between levels A, B, and C — indicating the dynamic change in access rights Kwaku

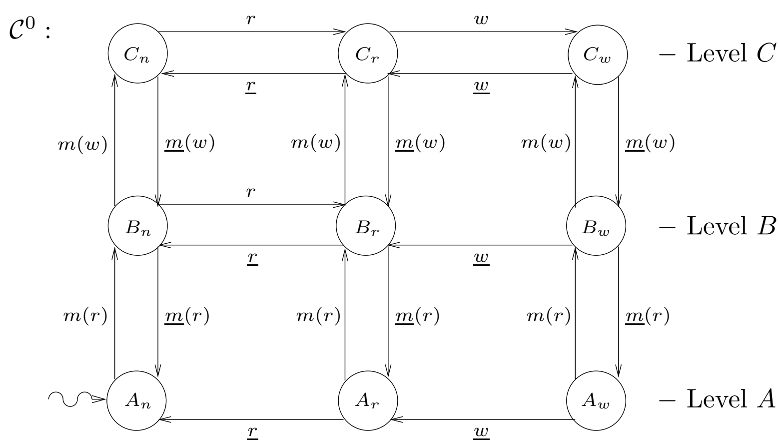


Fig. 8.14. Component automaton $\mathcal{C}^0$: the access building.

has for F — and horizontally between the states "null", "reading", and "writing" — indicating the current activities of Kwaku with respect to F. Notice that in $\mathcal{C}^0$, e.g., the state B_w meaning that Kwaku is writing while having read access but no write access, can only be reached from C_w by an action $\underline{m}(w)$ or from A_w by an action $m(r)$. Hence this state B_w can be entered only when Kwaku is writing while his status changes. There is no transition to B_w at level B. A similar remark holds for states A_r and A_w, which can be entered only from level B by the read access revocation action $\underline{m}(r)$. States such as A_r, A_w, and B_w are called *irregular states* because they are not reachable at their own level.

To model meta access control, we assume the existence of a system administrator, Abena, who can change Kwaku's rights. Hence Abena has the right to grant and revoke access by Kwaku to F. For this reason we have chosen all actions of granting and revoking access rights in $\mathcal{C}^0$ to be input actions, while all actions of reading and writing are output actions. The right to grant and revoke are legitimate rights, but they are not directly applied to F. They are in fact meta operations — hence $m(r)$ and $m(w)$ — and the rights to apply these meta operations are meta rights. Similarly, if there is a creator, Kwesi, who can allow (and disallow) Abena to grant and revoke, then Kwesi has meta meta rights. Kwesi has the meta meta right to grant and revoke Abena's meta rights to grant and revoke Kwaku's access rights to F. A typical action of Kwesi is $\underline{m}^2(w)$, which revokes Abena's right to grant and revoke write access to Kwaku.

The notion of meta clearly extends to arbitrary layers. An example of such a multi-layered structure of meta can be seen in the journal refereeing process. The creator of a document may delegate publication responsibilities to co-authors who may select a journal and grant $m^2(r)$ rights to the editor-in-chief. The editor-in-chief may grant $m(r)$ rights to assistant editors who can then grant and revoke read access to reviewers. An interesting question now arises as to the effect of revocation: should revocation of a meta right also revoke the rights that were passed on to others? This is the issue of *shallow revocation* versus *deep revocation*. Shallow revocation means that a revoke action does not revoke any of the rights that were previously passed on to others, whereas deep revocation means that a revoke action does revoke all rights previously passed on. Team automata can be used to model shallow, deep, or even hybrid revocation. Shallow revocation is often the easiest to model, whereas deep revocation is known as a big challenge to model and implement ([DS98]). We now show how deep revocation can be modeled using team automata.

Figure 8.15 shows a component automaton capturing one layer (layer k) of a multi-layer meta access specification for our example of read and write access. We have already seen layer 0, viz. component automaton $\mathcal{C}^0$. For each value of $k \geq 1$ there are corresponding component automata that are directly related to layer k (viz. $\mathcal{C}^{k-1}$ at layer $k-1$ and $\mathcal{C}^{k+1}$ at layer $k+1$). For each such component automaton $\mathcal{C}^k$, the horizontal actions $m^k(r)$, $\underline{m}^k(r)$, $m^k(w)$, and $\underline{m}^k(w)$ are output actions, whereas the vertical actions $m^{k+1}(r)$, $\underline{m}^{k+1}(r)$, $m^{k+1}(w)$, and $\underline{m}^{k+1}(w)$ are input actions. For $k = 0$ we identify r with $m^0(r)$, $\underline{r}$ with $\underline{m}^0(r)$, w with $m^0(w)$, and $\underline{w}$ with $\underline{m}^0(w)$. Similarly, $m(r) = m^1(r)$, $\underline{m}(r) = \underline{m}^1(r)$, $m(w) = m^1(w)$, and $\underline{m}(w) = \underline{m}^1(w)$.

We can now define a multi-layered structure by recursively composing a team automaton over $\mathcal{C}^0$, $\mathcal{C}^1$, $\ldots$, and $\mathcal{C}^n$, for some $n \geq k$. Note that $\{\mathcal{C}^0, \mathcal{C}^1, \ldots, \mathcal{C}^n\}$ is a composable system. As mentioned before we can also build this team automaton in an iterated way starting from, e.g., a team over any two component automata $\mathcal{C}^k$ and $\mathcal{C}^{k+1}$. In Figure 8.16, the state-reduced version $(\mathcal{T}^k_{k-1})_S$ of a team automaton $\mathcal{T}^k_{k-1}$ over $\mathcal{C}^{k-1}$ and $\mathcal{C}^k$, representing layer $k-1$ and layer k of this layered structure, is depicted.

The transition relation of this team $\mathcal{T}^k_{k-1}$ is chosen with the modeling of deep revocation in mind. Finally, note that in Figure 8.16 we have added superscripts to distinguish the states in $\mathcal{C}^k$ from the states in $\mathcal{C}^{k-1}$, e.g., state B_r of $\mathcal{C}^k$ from state B_r of $\mathcal{C}^{k-1}$.

In our example, $\mathcal{C}^2$ represents the actions of the supervisor Kwesi and $\mathcal{C}^1$ those of Abena. Now consider Kwesi in state B^2_r. Then Figure 8.16 tells us that Abena must be in one of the three states B^1_n, B^1_r, or B^1_w. Assume

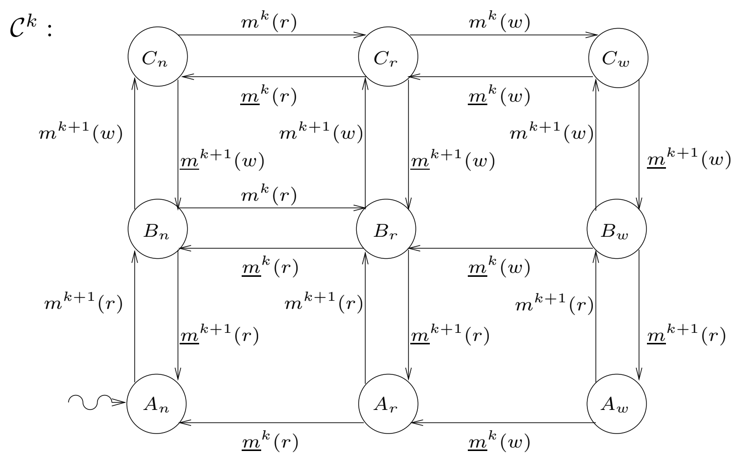


Fig. 8.15. Component automaton $\mathcal{C}^k$: meta access at layer k.

that Kwesi reached this state B_r^2 by performing action $m^2(r)$ from B_n^2, while Abena was in state A_n^1 having no rights to grant and revoke reading rights. Action $m^2(r)$ is an output action of $\mathcal{C}^2$ and an input action of $\mathcal{C}^1$, and our transition relation forces $\mathcal{C}^1$ to transition from A_n^1 to B_n^1. The interpretation is that Kwesi granted Abena the right to do read grants and revokes (to user Kwaku for file F).

Similarly, component automaton $\mathcal{C}^k$ can revoke the right to grant and to revoke read access from $\mathcal{C}^{k-1}$ at any time by performing output action $\underline{m}^k(r)$, and thus forcing $\mathcal{C}^{k-1}$ to perform this action — this time as an input action — as well. Continuing our example, this means that while in state B_r^2, Kwesi's read granting right may be revoked by action $\underline{m}^3(r)$ at any time. If this happens, Kwesi is forced into the irregular state A_r^2, which has only one possible output action, viz. $\underline{m}^2(r)$, leading to A_n^2. Whenever that action $\underline{m}^2(r)$ occurs it revokes Abena's right to change Kwaku's read access.

We thus observe two general rules of activity in such a team automaton over $\{\mathcal{C}^0, \mathcal{C}^1, \ldots, \mathcal{C}^n\}$, with each component automaton of the form depicted in Figure 8.15. First, when a "master" component automaton $\mathcal{C}^k$ where $1 \leq k \leq n$, transitions right (grant) or left (revoke), then the "slave" component automaton $\mathcal{C}^{k-1}$ must transition upward (gaining some access right) or downward (losing some access right). Secondly, the slave $\mathcal{C}^{k-1}$ may be forced to transition downward into an irregular state, in which case it will eventually transition to the left. $\mathcal{C}^{k-1}$ is itself a master and thus this transition to the left again forces a downward transition of $\mathcal{C}^{k-2}$, and so on until $\mathcal{C}^0$ on layer 0. Hence, as promised, we indeed model deep revocation.

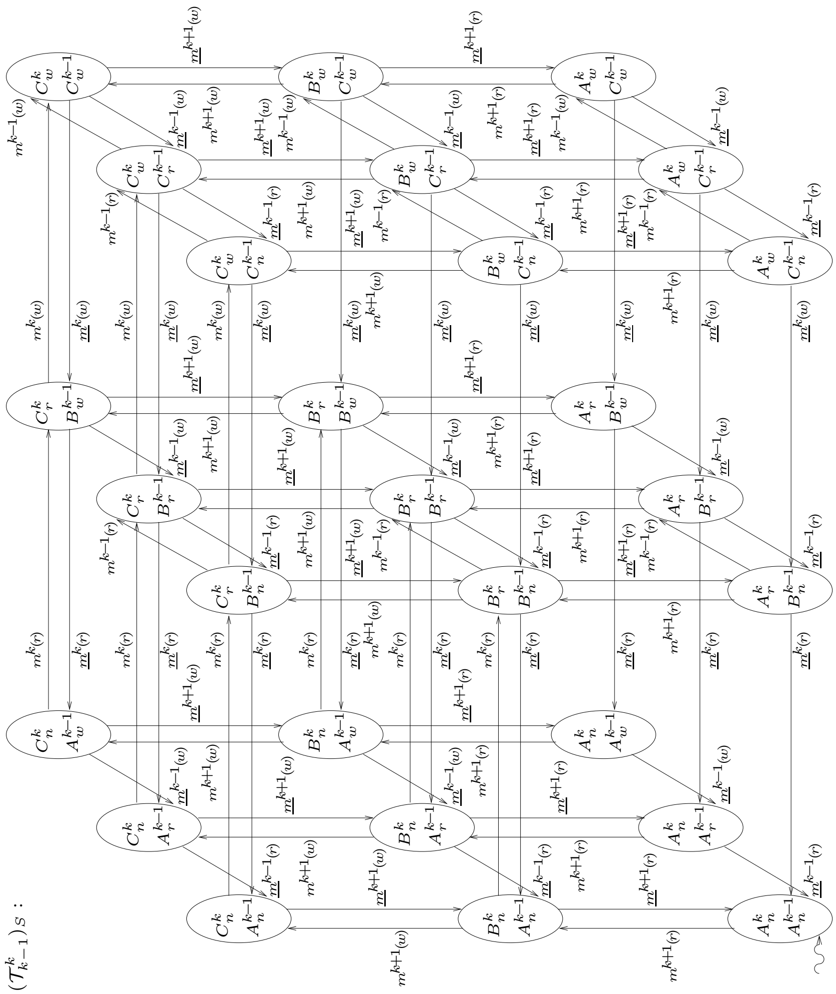


Fig. 8.16. State-reduced team automaton $(\mathcal{T}^k_{k-1})s$ over $\mathcal{C}^{k-1}$ and $\mathcal{C}^k$.

8.3.4 Conclusion

In this section we have demonstrated by means of examples how team automata can be used for modeling access control mechanisms presented through the metaphor of spatial access. The combination of the formal framework of team automata and the spatial access metaphor leads to a powerful abstraction well suited for a precise description of (at least some of the) key issues of access control. The team automata framework supports the design of distributed systems and protocols, by making explicit the role of actions and the choice of transitions governing the communication, coordination, cooperation, and collaboration. Examples include, e.g., peer-to-peer and master-slave types of synchronization, or heterogenous combinations thereof. Moreover, the formal setup and the possibility of a modular design provide analytic tools for the verification of desired properties of complex (computer) systems. Team automata are thus a fitting companion to the virtual spaces metaphor used in virtual reality systems that supports notions of rooms and buildings. Each space is represented by a component automaton, dynamic access changes are represented by joint external actions, while resource accesses within a space can be represented by internal actions.

Obviously there are numerous other possible examples as well as variations of the example we have considered above. For one, the assumption that write access can only be granted if read access has been granted can easily be dropped. Similarly, grant and revoke rights can be coupled more loosely. Read and write operations are specified here at the file level, but could also have been specified at the page level, object level, or record level, to name but a few. This might mean that delayed revocation is precisely the right choice. At the file level, the r and $\underline{r}$ actions might be seen at the user interface as open and close file. The w and $\underline{w}$ actions might be edit and save operations. When dealing with a transaction system, combinations of these operations might correspond to begin transaction and end transaction.

The team automata framework handles group decision making well and therefore allows convenient implementations of *distributed access control*. Distributed access control means that the supervisory work of granting and revoking access rights is administered by multiple agents. Thus Kwaku could have two administrative supervisors who must agree on any change of access rights. This can be modeled as an action of two masters and one slave: the actions would be output for both supervisors, requiring both to participate, and input for the slave. Alternatively, by including transitions with one supervisor being inactive, we can model the case of approval being required by either one of the two supervisors. Hybrids between pure master-slave and pure peer-to-peer types of synchronization, as in heterogenous team automata, are also useful. All these variations are due to the fact that the choice of a transition relation is the crucial modeling issue of the team automata framework.

Recall that team automata model the logical architecture of a design. They abstract from concrete data, configurations, and actions, and only describe behavior in terms of a state-action diagram (structure), the role of actions (input, output, or internal), and synchronizations (shared actions). It is not feasible (nor necessary) to have a distinct component automaton for each individual, and for each file in an organization. In many situations, categories and roles are used rather than individuals. Any implementation would have the team automaton as a class entity, and an activation record for each person, containing their current state. Similarly, by keeping a status of the files one can model the criterion "only one person can write a file at a time, but many readers is OK". The model cast in the spirit of component automata depicting roles rather than individuals becomes much more useful and general, and avoids some notational problems of exponential growth.

As observed earlier, time and priorities are not incorporated in neither the spatial access metaphor nor the team automata model as discussed here. However, similar to the Petri net model one may consider to extend team automata with time and priorities (see, e.g., [ABC$^+$95], which focuses on performance analysis). When time and/or priorities are part of access control this would allow the designer to control the sojourn times in the local states and to control the resolution of conflicting actions.

Using team automata for modeling (spatial) access control forces one to make explicit and unambiguous design choices and at the same time provides the possibility of mathematically precise analysis tools for proving crucial design properties, without first having to implement one's design.

9. Discussion

In this chapter we summarize the main contributions of this thesis and point out some topics worth further investigation. We moreover indicate how — in theory — team automata can be used for system design and where — in practice — they have actually been used.

Contributions of the Thesis

In this thesis we have formally presented team automata as a model for component-based system design. Team automata are based on the well-known method for modeling collaboration between system components by synchronizations of actions or transitions. A distinguishing feature of team automata is the freedom to choose on which actions and when their constituting component automata synchronize. In addition, there is the distinction of a team automaton's alphabet into input, output, and internal actions.

Through the classification of a broad range of ways to synchronize actions in team automata, a systematic study of the role that synchronizations play when modeling collaboration between system components has been conducted. To begin with, we have studied their effect on the inheritance of various automata-theoretic properties from team automata to their constituting component automata and subteams, and vice versa. We have furthermore studied their effect on the inheritance of various automata-theoretic properties from team automata to their constituting component automata and subteams, and vice versa. These studies are not complete and thus offer interesting pointers for further investigation.

The relation between team automata and two related models, viz. I/O automata and Petri nets, has been investigated in considerable detail. This has shown that I/O automata fit into the framework of team automata, whereas so-called non-state-sharing vector team automata can be translated into ITNCs — a model of vector-labeled Petri nets. Vector team automata are team automata in which the (team) actions have been replaced by vectors of (component) actions, from which the participation of a component automaton in a synchronization can thus be seen immediately. Consequently, non-state-sharing vector team automata are the subclass of vector team automata with the characteristic that whether or not a synchronization can take place only depends on the local states of the component automata actively involved in that synchronization. As a result, synchronizations involving disjoint sets of component automata are independent, which would thus allow a concurrent semantics for non-state-sharing vector team automata. This is a point worth further investigation.

Team automata are naturally suited for component-based system design due to the fact that they can themselves be used as component automata of higher-level team automata. This allows the iterative composition of team automata. We have been able to show that iterated composition does not lead to an increase of the number of possibilities for synchronization. Every iterated team automaton over a composable system can be interpreted as a team automaton over that composable system, by reordering its state space and transition space. We have moreover been able to show that every team automaton can be iteratively composed over its subteams.

We have studied the computations and behavior of team automata in relation to those of their constituting component automata. Several types of team automata that satisfy compositionality could be identified. To describe the compositionality of team automata, we have had to develop an extensive theory of (synchronized) shuffles. An examiniation of the compositionality of further types of team automata is certainly a topic worth further investigation. This might very well require the introduction and analysis of more sophisticated types of shuffles.

Using Team Automata

Modeling a system as a team automaton in the early phases of design forces one to identify the active components of the system and to consider the intended communications and synchronizations in detail, which is bound to lead to a better understanding of system functionality and to explicit and unambiguous design choices. This forms the basis of further design and implementation, while at the same time the mathematically rigorous definitions provide the possibility of formal analysis tools for proving crucial design properties, without first having to implement the design.

In Theory

To model a system as a team automaton, first the components have to be identified. Each of them should be given a description in the form of an automaton — an easy to understand model that moreover forms the basis for system descriptions in a number of model-checking tools (see, e.g., [Hol91], [Kur94], [Hol97], and [Hol03]). Based on the idea of synchronizations of common actions, these components can be connected in order to collaborate. Within each component, a distinction has to be made between internal actions — which are not available for synchronization with other components — and external actions — which can be used to synchronize components and may be subject to synchronization restrictions. By assigning such different roles to actions it is possible to describe many types of collaboration.

Consequently, for each external action separately, a decision is made as to how and when the components should synchronize on this action. If the action is supposed to be a passive action that may not be under the component's local control, then it can be designated as an input action of that component, otherwise as an output action. If such a distinction between the roles of an external action is not necessary, then the choice is arbitrary. A natural option would be to make it an output action in all components in which it occurs. Once the synchronization constraints for each external action have been determined, one may apply, e.g., a maximality principle to construct a unique team automaton satisfying all constraints.

The team automata framework thus supports component-based system design by making explicit the role of actions and the choice of transitions that govern the collaboration between components. The crucial feature is the freedom of choice for the synchronizations collected in the transition relation of a team automaton. This is indeed one of the main reasons given in [Ell97] for introducing team automata to model groupware systems rather than using I/O automata for that purpose. Another important reason is that, in order for a team automaton to be capable of modeling various types of collaboration between its components by synchronizations of common actions, synchronizations between output actions of its components should not be excluded a priori. As a matter of fact, the peer-to-peer types of synchronization explicitly use the possibility to synchronize on output actions. Finally, no matter how convenient input enabling may be when modeling reactive systems, it does hinder a realistic modeling of collaborations that involve humans — in fact, Tuttle himself was the first to acknowledge this when he introduced I/O automata in [Tut87] (cf. Section 7.1) — while modeling such collaborations was one of the main reasons for the introduction of team automata.

In Practice

An increasing number of papers bears witness to the usefulness of team automata in the early design phase of reactive systems in general, and of groupware systems in particular. Moreover, these examples are not limited to modeling within CSCW (see, e.g., [Ell97], [EK00], [Lav00], [BEKR01a], [BEKR01b], and [BB03]) but extend to areas such as software engineering (see, e.g., [HB00], [Hoe01], and [EG02]) and — most recently — security (see, e.g., [BLP03]). In fact, a spectrum from hardware components to protocols for interacting groups of people has been modeled by team automata. There is still quite some work left to do, though. For one, the components of a team currently cannot exchange any information, i.e. they have no private memory. In order to be useful also in later stages of the design of groupware systems (or to model, e.g., workflow systems) team automata should thus — among other things — be extended with the flow of information between components. An initial attempt in this direction was recently undertaken in [BCM03]. Furthermore, team automata are currently inappropriate for capturing aspects of group activity such as social aspects and informal unstructured activity.

We now close this Discussion with an initial observation on the potential of team automata within a process model recently introduced in the field of CSCW. In [Dew01], Dewan claims that traditional software process models such as the waterfall model and the spiral model — while efficient for describing the different phases in the life cycle of software in general — lack too many "collaboration-specific details" to be efficient for "collaborative systems". These are software systems including "both general infrastructures and specific applications for supporting collaboration". Therefore, Dewan proposes a new process model well suited for collaborative systems.

The initial phase of Dewan's model consists of decomposing the functionality of collaborative systems into smaller subfunctions, which can be worked upon more-or-less independently. Examples of such collaboration functions are listed in [DCS94] and [Dew01]. Among them are *merging* and *access control*. Merging combines independent versions into a single object, whereas access control determines the operations a user is authorized to perform. In Section 8.2 we showed how team automata could be applied — in a conflict-free strategy — to merge previously distributed packages back together. In Section 8.3 we consequently showed how access control mechanisms could be made precise and given a formal description using team automata. Team automata thus seem promising for modeling these two subfunctions of Dewan's process model.

Bibliography

[ABC$^+$95] M. Ajmone Marson, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with generalized stochastic Petri nets*, John Wiley & Sons, Chichester, 1995.

[Arn82] A. Arnold, Synchronized Behaviours of Processes and Rational Relations. *Acta Informatica* 17 (1982), 21 – 29.

[Arn94] A. Arnold, *Finite Transition Systems*, Prentice Hall International Series in Computer Science, London, 1994.

[AN82] A. Arnold and M. Nivat, Comportements de processus. In *Colloque AFCET Les Mathématiques de l'Informatique*, 1982, 35 – 68. (*In French.*)

[BDQT99] E. Badouel, Ph. Darondeau, D. Quichaud, and A. Tokmakoff, Modelling Dynamic Agent Systems with Cooperating Automata. Publication Interne 1253, Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes, 1999.

[BB03] M.H. ter Beek and R.P. Bloem, Model Checking Team Automata for Access Control. Unpublished manuscript, 2003.

[BCM03] M.H. ter Beek, E. Csuhaj-Varjú, and V. Mitrana, Teams of Pushdown Automata. To appear in *Proceedings of the PSI'03 Fifth International Conference on Perspectives of System Informatics, Novosibirsk, Akademgorodok, Russia* (A. Zamulin and M. Broy, eds.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2003. (A full version appeared as Technical Report 2002/4, Research Group on Modelling Multi-Agent Systems, Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, 2002.)

[BEKR01a] M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, Team Automata for CSCW. In *Proceedings of the Second International Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin, Germany* (H. Weber, H. Ehrig, and W. Reisig, eds.), Fraunhofer Institute for Software and Systems Engineering, Berlin, 2001, 1 – 20. (Also appeared as Technical Report TR-01-07, Leiden Institute of Advanced Computer Science, Universiteit Leiden, Leiden, 2001.)

[BEKR01b] M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, Team Automata for Spatial Access Control. In *Proceedings of the ECSCW'01 Seventh European Conference on Computer Supported Cooperative Work, Bonn, Germany* (W. Prinz, M. Jarke, Y. Rogers, K. Schmidt, and V. Wulf, eds.), Kluwer Academic Publishers, Dordrecht, 2001, 59 – 77.

[BEKR03] M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, Synchronizations in team automata for groupware systems. *Computer Supported*

Cooperative Work — The Journal of Collaborative Computing 12, 1 (2003), 21 – 69.

[BK03] M.H. ter Beek and J. Kleijn, Team Automata Satisfying Compositionality. In *Proceedings of FME 2003: Formal Methods — the Twelfth International Symposium of Formal Methods Europe, Pisa, Italy* (K. Araki, S. Gnesi, and D. Mandrioli, eds.), *Lecture Notes in Computer Science* 2805, Springer-Verlag, Berlin, 2003, 381 – 400.

[BLP03] M.H. ter Beek, G. Lenzini, and M. Petrocchi, Team Automata for Security Analysis of Multicast/Broadcast Communication. In *Proceedings of the WISP'03 Workshop on Issues in Security and Petri Nets, Eindhoven, The Netherlands* (N. Busi, R. Gorrieri and F. Martinelli, eds.), Beta Research School for Operations Management and Logistics, Department of Technology Management, Eindhoven University of Technology, Eindhoven, 2003, 57 – 71. (Also appeared as Technical Report 2003-TR-13, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, 2003.)

[BPS01] *Handbook of Process Algebra* (J.A. Bergstra, A. Ponse, and S.A. Smolka, eds.), Elsevier Science Publishers, Amsterdam, 2001.

[BC92] L. Bernardinello and F. De Cindio, A Survey of Basic Net Models and Modular Net Classes. In *Advances in Petri Nets 1992* (G. Rozenberg, ed.), *Lecture Notes in Computer Science* 609, Springer-Verlag, Berlin, 1992, 304 – 351.

[BÉ96] S.L. Bloom and Z. Ésik, Free Shuffle Algebras in Language Varieties. *Theoretical Computer Science* 163 (1996), 55 – 98.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM* 31, 3 (1984), 560 – 599.

[Bul98] A. Bullock, *SPACE: Spatial Access Control in Collaborative Virtual Environments*. Ph.D. thesis, Department of Computer Science, University of Nottingham, 1998.

[BB97] A. Bullock and S. Benford, Access Control in Virtual Environments. In *Proceedings of the VRST'97 ACM Symposium on Virtual Reality Software and Technology, Lausanne, Switzerland* (D. Thalmann, S. Feiner, and G. Singh, eds.), ACM Press, New York, 1997, 29 – 35.

[BB99] A. Bullock and S. Benford, An access control framework for multi-user collaborative environments. In *Proceedings of the GROUP'99 International ACM SIGGROUP Conference on Supporting Group Work, Phoenix, Arizona*, ACM Press, New york, 1999, 140 – 149.

[CH74] R.H. Campbell and A.N. Habermann, The Specification of Process Synchronisation by Path Expressions. In *Proceedings of an International Symposium on Operating Systems, Rocquencourt, France* (E. Gelenbe and C. Kaiser, eds.), *Lecture Notes in Computer Science* 16, Springer-Verlag, Berlin, 1974, 89 – 102.

[CC02] J. Carmona and J. Cortadella, Input/Output Compatibility of Reactive Systems. In *Fourth International Conference on Formal Methods in Computer-Aided Design, Portland, Oregon* (M.D. Aagaard and J.W. O'Leary, eds.), *Lecture Notes in Computer Science* 2517, Springer-Verlag, Berlin, 2002, 360 – 377.

[CCP02] J. Carmona, J. Cortadella, and E. Pastor, Synthesis of Reactive Systems: Application to Asynchronous Circuit Design. In *Concurrency and Hardware Design — Advances in Petri Nets* (J. Cortadella, A. Yakovlev, and G. Rozenberg, eds.), Springer-Verlag, Berlin, 2002, 107 – 151.

[CW96] E.M. Clarke and J.M. Wing, Formal methods: State of the art and future directions. *ACM Computing Surveys* 28, 4 (1996), 626 – 643.

[Dar91] S. Dart, Concepts in Configuration Management Systems. In *Proceedings of the Third International Workshop on Software Configuration Management, Trondheim, Norway* (P.H. Feiler, ed.), ACM Press, New York, 1991, 1 – 18.

[DKW99] *Software Process: Principles, Methodology, Technology* (J.-C. Derniame, A.B. Kaba, and D. Wastell, eds.), *Lecture Notes in Computer Science* 1500, Springer-Verlag, Berlin, 1999.

[DeS84] R. De Simone, Langages Infinitaires et Produit de Mixage. *Theoretical Computer Science* 31 (1984), 83 – 100.

[Dew01] P. Dewan, An integrated Approach to Designing and Evaluating Collaborative Applications and Infrastructures. *Computer Supported Cooperative Work — The Journal of Collaborative Computing* 10, 1 (2001), 75 – 111.

[DCS94] P. Dewan, R. Choudhary, and H. Shen, An Editing-Based Characterization of the Design Space of Collaborative Applications. *Journal of Organizational Computing* 4, 3 (1994), 219 – 240.

[DS98] P. Dewan and H. Shen, Flexible Meta Access-Control for Collaborative Applications. In *Proceedings of the CSCW'98 ACM Conference on Computer Supported Cooperative Work, Seattle, Washington* (E. Churchill, D. Snowdon, and G. Golovchinsky, eds.), ACM Press, New York, 1998, 247 – 256.

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

[DH94] D. Drusinsky and D. Harel, On the Power of Bounded Concurrency I: Finite Automata. *Journal of the ACM* 41, 3 (1994), 517 – 539.

[Dub86] C. Duboc, Mixed Product and Asynchronous Automata. *Theoretical Computer Science* 42 (1986), 183 – 199.

[Ell97] C.A. Ellis, Team Automata for Groupware Systems. In *Proceedings of the GROUP'97 International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge, Phoenix, Arizona* (S.C. Hayne and W. Prinz, eds.), ACM Press, New York, 1997, 415 – 424.

[EGR90] C.A. Ellis, S.J. Gibbs, and G. Rein, Design and Use of a Group Editor. In *Engineering for Human Computer Interaction* (G. Cockton, ed.), North-Holland Publishing Company, Amsterdam, 1990, 13 – 25.

[EK00] C.A. Ellis and K.-H. Kim, A Framework and Taxonomy for Workflow Architecture. In *Designing Cooperative Systems: The Use of Theories and Models — Proceedings of the COOP'2000 International Conference on the Design of Cooperative Systems, Sophia Antipolis, France* (R. Dieng, A. Giboin, L. Karsenty, and G. De Michelis, eds.), *Frontiers in Artificial Intelligence and Applications* 58, IOS Press, Amsterdam, 2000, 99 – 112.

[EN93] C.A. Ellis and G.J. Nutt, Modelling and Enactment of Workflow Systems. In *Proceedings of the ATPN'93 International Conference on Application and Theory of Petri Nets, Chicago, Illinois* (M. Ajmone Marsan, ed.), *Lecture Notes in Computer Science* 691, Springer-Verlag, Berlin, 1993, 1 – 16.

[EG02] G. Engels and L.P.J. Groenewegen, Towards Team-Automata-Driven Object-Oriented Collaborative Work. In *Formal and Natural Computing - Essays Dedicated to Grzegorz Rozenberg* (W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, eds.), *Lecture Notes in Computer Science* 2300 (2002), 257 – 276.

[GSSL94] R. Gawlick, R. Segala, F.F. Søgaard-Andersen, and N. Lynch, Liveness in Timed and Untimed Systems. In *Proceedings of the ICALP'94 Twenty-first International Colloquium on Automata, Languages and Programming, Jerusalem, Israel* (S. Abiteboul and E. Shamir, eds.), *Lecture Notes in Computer Science* 820, Springer-Verlag, Berlin, 1994, 166 – 177. (A full version appeared as Technical Report MIT/LCS/TR-587, Massachusetts Institute of Technology, Cambridge, Massachusetts.)

[GS65] S. Ginsburg and E.H. Spanier, Mappings of Languages by Two-Tape Devices. *Journal of the ACM* 12, 3 (1965), 423 – 434.

[Gis81] J.L. Gischer, Shuffle Languages, Petri Nets, and Context Sensitive Grammars. *Communications of the ACM* 24 (1981), 597 – 605.

[Gru94] J. Grudin, CSCW: History and Focus. *IEEE Computer* 27, 5 (1994), 19 – 26.

[Har87] D. Harel, Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming* 8 (1987), 231 – 274.

[HH94] T. Hirst and D. Harel, On the Power of Bounded Concurrency II: Pushdown Automata. *Journal of the ACM* 41, 3 (1994), 540 – 554.

[Hoa78] C.A.R. Hoare, Communicating Sequential Processes. *Communications of the ACM* 21, 8 (1978), 666 – 677.

[Hoa85] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall International Series in Computer Science, London, 1985.

[Hoe01] P.J. 't Hoen, *Towards Distributed Development of Large Object-Oriented Models — Views of Packages as Classes*. Ph.D. thesis, Leiden Institute of Advanced Computer Science, Leiden University, 2001.

[HB00] P.J. 't Hoen and M.H. ter Beek, A Conflict-Free Strategy for Team-Based Model Development. In *Proceeedings of the PDTSD'00 International Workshop on Process support for Distributed Team-based Software Development in Volume IX:Industrial Systems of the Proceedings of the SCI'00 World MultiConference on Systemics, Cybernetics and Informatics, Orlando, Florida* (B. Sanchez, R. Hammel II, M. Soriano, and P. Tiako, eds.), International Institute of Informatics and Systemics, 2000, 720 – 725.

[Hol91] G.J. Holzmann, *Design and Validation of Computer Protocols*, Prentice Hall International, Inc., Englewood Cliffs, New Jersey, 1991.

[Hol97] G.J. Holzmann, The model checker SPIN. *IEEE Transactions on Software Engineering* 23, 5 (1997), 279 – 295.

[Hol03] G.J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, Addison Wesley Publishers, Reading, Massachusetts, 2003.

[IEEE93] ANSI/IEEE Standard 1042-1987, *IEEE Guide to Software Configuration Management. IEEE Standards Collection — Software Engineering, 1993 Edition*, Institute of Electrical and Electronics Engineers, Inc., New York, 1993.

[JL92] R. Janicki and P.E. Laurer, *Specification and Analysis of Concurrent Systems, The COSY Approach. EATCS Monograhps on Theoretical Computer Science*, Springer-Verlag, Berlin, 1992.

[Jan81] M. Jantzen, The Power of Synchronizing Operations on Strings. *Theoretical Computer Science* 14 (1981), 127 – 154.

[Jon87] B. Jonsson, Compositional Verification of Distributed Systems. Ph.D. thesis, Department of Computer Systems, Uppsala University, 1987.

[Jon94] B. Jonsson, Compositional Specification and Verification of Distributed Systems. *ACM Transactions on Programming Languages and Systems* 16, 2 (1994), 259 – 303.

[Kee96] N.W. Keesmaat, *Vector Controlled Concurrent Systems*. Ph.D. thesis, Department of Computer Science, Leiden University, 1996.

[KK97] N.W. Keesmaat and H.C.M. Kleijn, Net-based Control versus Rational Control: The Relation between ITNC Vector Languages and Rational Relations. *Acta Informatica* 34 (1997), 23 – 57.

[KKR90] N.W. Keesmaat, H.C.M. Kleijn, and G. Rozenberg, Vector Controlled Concurrent Systems, Part I: Basic Classes. *Fundamenta Informaticae* 13 (1990), 275 – 316.

[KKR91] N.W. Keesmaat, H.C.M. Kleijn, and G. Rozenberg, Vector Controlled Concurrent Systems, Part II: Comparisons. *Fundamenta Informaticae* 14 (1991), 1 – 38.

[KB95] S. Khoshafian and M. Buckiewicz, *Introduction to Groupware, Workflow, and Workgroup Computing*, John Wiley & Sons, New York, 1995.

[Kim76] T. Kimura, An Algebraic System for Process Structuring and Interprocess Communication. In *Proceedings of the Eighth ACM SIGACT Symposium on Theory of Computing, Hershey, Pennsylvania*, ACM Press, New York, 1976, 92 – 100.

[Kur94] R.P. Kurshan, *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*, Princeton University Press, Princeton, New Jersey, 1994.

[LMP00] R. Lanotte, A. Maggiolo-Schettini, and A. Peron, Timed Cooperating Automata. *Fundamenta Informaticae* 42 (2000), 1 – 21.

[LR99] M. Latteux and Y. Roos, Synchronized Shuffle and Regular Languages. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa* (J. Karhumäki, H.A. Maurer, Gh. Păun, and G. Rozenberg, eds.), Springer-Verlag, Berlin, 1999, 35 – 44.

[LTS79] P.E. Laurer, P.R. Torregiani, and M.W. Shields, COSY — A System Specification Language based on Paths and Processes. *Acta Informatica* 12 (1979), 109 – 158.

[Lav00] H. Lavana, *A Universally Configurable Architecture for Taskflow-Oriented Design of a Distributed Collaborative Computing Environ-*

ment. Ph.D. thesis, Department of Electrical and Computer Engineering, North Carolina State University, 2000.

[Lyn96] N.A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, San Mateo, California, 1996.

[LT87] N.A. Lynch and M.R. Tuttle, Hierarchical Correctness Proofs for Distributed Algorithms. In *Proceedings of the Sixth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Vancouver, British Colombia, Canada*, 1987, 137 – 151.

[LT89] N.A. Lynch and M.R. Tuttle, An Introduction to Input/Output Automata. *CWI Quarterly* 2, 3 (1989), 219 – 246. (Also appeared as Technical Memo MIT/LCS/TM-373, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1988.)

[Maz89] A. Mazurkiewicz, Basic Notions of Trace Theory. In *Lecture Notes in Computer Science* 354, Springer-Verlag, Berlin, 1989, 285 – 363.

[Mey92] B. Meyer, Applying Design by Contract. *IEEE Computer* 25, 10 (1992), 40 – 51.

[Mil80] R. Milner, *A Calculus of Communicating Systems*, *Lecture Notes in Computer Science* 92, Springer-Verlag, Berlin, 1980.

[Mil89] R. Milner, *Communication and Concurrency*, Prentice Hall International Series in Computer Science, London, 1989.

[Niv79] M. Nivat, Sur la synchronisation des processus. *Revue Technique Thomson-CSF* 11 (1979), 899 – 919. (*In French.*)

[Ohe03] D. von Oheimb, Interacting State Machines: A Stateful Approach to Proving Security. To appear in *Proceedings of the BCS-FACS International Conference on Formal Aspects of Security* (A. Abdallah, P. Ryan, and S. Schneider, eds.), *Lecture Notes in Computer Science* 2629, Springer-Verlag, 2003.

[OL02] D. von Oheimb and V. Lotz, Formal Security Analysis with Interacting State Machines. In *Proceedings of the Seventh ESORICS'02 European Symposium on Research in Computer Security* (D. Gollmann, G. Karjoth, M. Waidner, eds.), *Lecture Notes in Computer Science* 2502, Springer-Verlag, 2002, 212 – 228.

[Par79] D. Park, On the Semantics of fair parallelism. In *Lecture Notes in Computer Science* 86, Springer-Verlag, Berlin, 1979, 504 – 526.

[Pet62] C.A. Petri, *Kommunikation mit Automaten*. Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Schrift Nr. 2, 1962. (*In German.*)

[RR98a] *Lectures on Petri Nets I: Basic Models* (W. Reisig and G. Rozenberg, eds.), *Lecture Notes in Computer Science* 1491, Springer-Verlag, Berlin, 1998.

[RR98b] *Lectures on Petri Nets II: Applications* (W. Reisig and G. Rozenberg, eds.), *Lecture Notes in Computer Science* 1492, Springer-Verlag, Berlin, 1998.

[Rod96] T. Rodden, Populating the Application: A Model of Awareness for Cooperative Applications. In *Proceedings of the CSCW'96 ACM Conference on Computer Supported Cooperative Work, Boston, Massachusetts* (M. Ackerman, ed.), ACM Press, New York, 1996, 87 – 96.

[Ros97] A.W. Roscoe, *The Theory and Practice of Concurrency*, Prentice Hall International Series in Computer Science, London, 1997.

[RS97] *Handbook of Formal Languages* (G. Rozenberg and A. Salomaa, eds.), Springer-Verlag, Berlin, 1997.

[RBP$^+$91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall International, Inc., Englewood Cliffs, New Jersey, 1991.

[Sha78] A.C. Shaw, Software Descriptions with Flow Expressions. *IEEE Transactions on Software Engineering* SE-4, 3 (1978), 242 – 254.

[SD92] H. Shen and P. Dewan, Access Control for Collaborative Environments. In *Proceedings of the CSCW'92 ACM Conference on Computer Supported Cooperative Work, Toronto, Canada* (J. Turner and R. Kraut, eds.), ACM Press, New York, 1992, 51 – 58.

[Shi79] M.W. Shields, Adequate Path Expressions. In *Proceedings of the Symposium on the Semantics of Concurrent Computation, Evian, France* (G. Kahn, ed.), *Lecture Notes in Computer Science* 70, Springer-Verlag, Berlin, 1979, 249 – 265.

[Shi97] M.W. Shields, *Semantics of Parallelism — Non-Interleaving Representation of Behaviour*, Springer-Verlag, Berlin, 1997.

[Sik97] K. Sikkel, A Group-based Authorization Model for Cooperative Systems. In *Proceedings of the ECSCW'97 Fifth European Conference on Computer Supported Cooperative Work, Lancaster, UK* (J. Hughes, W. Prinz, T. Rodden, and K. Schmidt, eds.), Kluwer Academic Publishers, Dordrecht, 1997, 345 – 360.

[Smi94] J. Smith, *Collective Intelligence in Computer Based Collaboration — A Volume in the Computers, Cognition, and Work Series*, Lawrence Erlbaum Associates, Mahwah, New Jersey, 1994.

[vdS85] J.L.A. van de Snepscheut, *Trace Theory and VLSI Design*, *Lecture Notes in Computer Science* 200, Springer-Verlag, Berlin, 1985.

[TH98] P.S. Thiagarajan and J.G. Henriksen, Distributed Versions of Linear Temporal Logic: A Trace Perspective. In [RR98a] (1998), 643 – 681.

[Tut87] M.R. Tuttle, *Hierarchical Correctness Proofs for Distributed Algorithms*. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1987. (Also appeared as Technical Report MIT/LCS/TR-387, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1987.)

[UML99] *Unified Modeling Language 1.3*, Technical Report, Rational Software Corporation, 1999.

[Zie87] W. Zielonka, Notes on Finite Asynchronous Automata. *RAIRO Informatique Théoretique et Applications* 21 (1987), 99 – 135.

List of Figures

List of Symbols

2. Preliminaries

$\subseteq$	set inclusion, 23
$\subset$	proper set inclusion, 23
$\setminus$	set difference, 23
$\#$	cardinality (of a set), 23
$\varnothing$	the empty set, 23
$[n]$	shorthand for $\{1, 2, \ldots, n\}$, 23
$\mathbb{N}$	set of positive integers, 23
$\prod$	cartesian product (prefix notation), 23
$\times$	cartesian product (infix notation), 23
proj_j	projection on element j, 23
proj_J	projection on subset J, 23
${\mathrm{proj}_j}^{[2]}$	shorthand for $\mathrm{proj}_j \times \mathrm{proj}_j$, 24
${\mathrm{proj}_J}^{[2]}$	shorthand for $\mathrm{proj}_J \times \mathrm{proj}_J$, 24
$f \upharpoonright C$	restriction of function f to a subset C of its domain, 24
Σ	alphabet, 24
λ	the empty word, 24
$\lvert w \rvert$	length (of a word w), 24
$w(i)$	i-th letter (of a word w), 24
$\#_a(w)$	total number of occurrences of letter a (in a word w), 24
$\mathrm{alph}(w)$	alphabet (of a word w), 25
Σ^*	set of all finite words over Σ, 25
Σ^+	set of all nonempty finite words over Σ, 25
Σ^ω	set of all infinite words over Σ, 25
Σ^∞	set of all words over Σ, 25
$u \cdot v$	concatenation (of words u and v), 25
$K \cdot L$	concatenation (of languages K and L), 25
$\mathrm{pref}(w)$	set of prefixes (of a word w), 26
$w[n]$	prefix of length n (of a word w), 25
$\lim\limits_{n\to\infty} v_n$	limit (of words $v_1 \leq v_2 \leq \cdots$), 26
pres_Γ	function preserving the symbols from Γ (and erasing all other symbols), 27

3. Automata

$\mathcal{A}$	automaton, 29
Q	set of states (of $\mathcal{A}$), 29
Σ	set of actions or alphabet (of $\mathcal{A}$), 29
δ	set of labeled transitions (of $\mathcal{A}$), 29
I	set of initial states (of $\mathcal{A}$), 29
δ_a	set of a-transitions (of $\mathcal{A}$), 30
$\mathbf{C}_{\mathcal{A}}$	set of finite computations of $\mathcal{A}$, 30
$\mathbf{C}_{\mathcal{A}}^{\omega}$	set of infinite computations of $\mathcal{A}$, 30
$\mathbf{C}_{\mathcal{A}}^{\infty}$	set of computations of $\mathcal{A}$, 30
$\mathbf{B}_{\mathcal{A}}^{\Theta,\infty}$	Θ-behavior of $\mathcal{A}$, 31
$\mathbf{B}_{\mathcal{A}}^{\Theta}$	finitary Θ-behavior of $\mathcal{A}$, 31
$\mathbf{B}_{\mathcal{A}}^{\Theta,\omega}$	infinitary Θ-behavior of $\mathcal{A}$, 31
Q_S	set of reachable states (of $\mathcal{A}$), 36
Σ_A	set of active actions (of $\mathcal{A}$), 36
δ_T	set of useful transitions (of $\mathcal{A}$), 36
$\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$	containment (of $\mathcal{A}_1$ in $\mathcal{A}_2$), 36
$\mathcal{A}_A^{\Theta}$	Θ-action-reduced version of $\mathcal{A}$, 37
$\mathcal{A}_T^{\Theta}$	Θ-transition-reduced version of $\mathcal{A}$, 38
$\mathcal{A}_S$	state-reduced version of $\mathcal{A}$, 46
$\mathcal{A}_A$	action-reduced version of $\mathcal{A}$, 50
$\mathcal{A}_T$	transition-reduced version of $\mathcal{A}$, 50
$\mathcal{A}_R$	reduced version of $\mathcal{A}$, 50

4. Synchronized Automata

$\mathcal{I}$	index set, 59
$\mathcal{A}_i$	automaton, 59
$\mathcal{S}$	set of automata, 59
$\Delta_a(\mathcal{S})$	complete transition space of a in $\mathcal{S}$, 60
$\mathcal{T}$	synchronized automaton, 60
$SUB_J(\mathcal{T})$	the subautomaton of $\mathcal{T}$ determined by J, 64
SUB_J	the subautomaton (of $\mathcal{T}$) determined by J, 64
$\pi_{\mathcal{A}_j}$	projection on automaton $\mathcal{A}_j$, 70
π_{SUB_J}	projection on subautomaton SUB_J, 70
$\mathcal{D}$	indexed set, 76
$\mathcal{V}(\mathcal{D})$	all finitely nested cartesian products of sets from $\mathcal{D}$, 76
$\mathrm{dom}\,(V)$	domain of an element V, 76
u_V	function unpacking elements v from V, 77

$\langle v \rangle_V$	reordering of an element $v \in V$ relative to the construction of V, 77
$\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}}$	reordered version of synchronized automaton $\mathcal{T}$ (w.r.t. $\mathcal{S}$), 81
$\mathcal{T}$	synchronized automaton, 84
$Free(\mathcal{T})$	set of *free* actions of $\mathcal{T}$, 85
$AI(\mathcal{T})$	set of *ai* actions of $\mathcal{T}$, 85
$SI(\mathcal{T})$	set of *si* actions of $\mathcal{T}$, 86
$\mathcal{R}_a^{no}(\mathcal{S})$	predicate *no-constraints*, 88
$\mathcal{R}_a^{free}(\mathcal{S})$	predicate *is-free* for a in $\mathcal{S}$, 88
$\mathcal{R}_a^{ai}(\mathcal{S})$	predicate *is-ai* for a in $\mathcal{S}$, 89
$\mathcal{R}_a^{si}(\mathcal{S})$	predicate *is-si* for a in $\mathcal{S}$, 89
j	element of $\mathcal{I}$, 90
J	subset of $\mathcal{I}$, 90
Θ	arbitrary alphabet disjoint from set Q of states (of $\mathcal{T}$), 90

5. Team Automata

$\mathcal{C}$	component automaton, 116
Σ_{inp}	set of input actions or input alphabet (of $\mathcal{C}$), 116
Σ_{out}	set of output actions or output alphabet (of $\mathcal{C}$), 116
Σ_{int}	set of internal actions or internal alphabet (of $\mathcal{C}$), 116
$\mathrm{und}(\mathcal{C})$	underlying automaton of $\mathcal{C}$, 116
Σ	set of actions or (full) alphabet (of $\mathcal{C}$), 116
Σ_{ext}	set of external actions or external alphabet (of $\mathcal{C}$), 116
Σ_{loc}	set of locally-controlled actions or locally-controlled alphabet (of $\mathcal{C}$), 117
$\mathbf{B}_{\mathcal{C}}^{\Sigma_{inp},\infty}$	input behavior (of $\mathcal{C}$), 117
$\mathbf{B}_{\mathcal{C}}^{\Sigma_{out},\infty}$	output behavior (of $\mathcal{C}$), 117
$\mathbf{B}_{\mathcal{C}}^{\Sigma_{int},\infty}$	internal behavior (of $\mathcal{C}$), 117
$\mathbf{B}_{\mathcal{C}}^{\Sigma_{ext},\infty}$	external behavior (of $\mathcal{C}$), 117
$\mathbf{B}_{\mathcal{C}}^{\Sigma_{loc},\infty}$	locally-controlled behavior (of $\mathcal{C}$), 117
$\mathcal{I}$	index set, 118
$\mathcal{C}_i$	component automaton, 118
Σ_i	set of actions (of $\mathcal{C}_i$), 118
$\mathcal{S}$	set of component automata, 118
$\mathcal{S}$	composable system, 118
$\mathcal{T}$	team automaton, 120
$\mathrm{und}(\mathcal{T})$	underlying synchronized automaton of $\mathcal{T}$, 120
$SUB_J(\mathcal{T})$	the subteam of $\mathcal{T}$ determined by J, 122
SUB_J	the subteam (of $\mathcal{T}$) determined by J, 122

$\mathcal{S}$	composable system, 123
$\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}}$	reordered version of team automaton $\mathcal{T}$ w.r.t. $\mathcal{S}$, 125
$\mathcal{T}$	team automaton, 126
Σ_{inp}	set of input actions (of $\mathcal{T}$), 126
Σ_{out}	set of output actions (of $\mathcal{T}$), 126
Σ_{int}	set of internal actions (of $\mathcal{T}$), 126
Σ	set of actions (of $\mathcal{T}$), 126
Σ_{ext}	set of external actions (of $\mathcal{T}$), 126
Σ_{loc}	set of locally-controlled actions (of $\mathcal{T}$), 126
$\mathcal{I}_{a,inp}(\mathcal{S})$	input domain of a in $\mathcal{S}$, 126
$\mathcal{I}_{a,out}(\mathcal{S})$	output domain of a in $\mathcal{S}$, 126
$\mathcal{I}_{a,inp}$	input domain of a (in $\mathcal{S}$), 127
$\mathcal{I}_{a,out}$	output domain of a (in $\mathcal{S}$), 127
$SUB_{a,inp}(\mathcal{T})$	input subteam of a in $\mathcal{T}$, 127
$SUB_{a,out}(\mathcal{T})$	output subteam of a in $\mathcal{T}$, 127
$SUB_{a,inp}$	input subteam of a (in $\mathcal{T}$), 127
$SUB_{a,out}$	output subteam of a (in $\mathcal{T}$), 127
$SIPP(\mathcal{T})$	set of *sipp* actions of $\mathcal{T}$, 129
$WIPP(\mathcal{T})$	set of *wipp* actions of $\mathcal{T}$, 129
$SOPP(\mathcal{T})$	set of *sopp* actions of $\mathcal{T}$, 129
$WOPP(\mathcal{T})$	set of *wopp* actions of $\mathcal{T}$, 129
$MS(\mathcal{T})$	set of *ms* actions of $\mathcal{T}$, 131
$SMS(\mathcal{T})$	set of *sms* actions of $\mathcal{T}$, 131
$WMS(\mathcal{T})$	set of *wms* actions of $\mathcal{T}$, 132
$\mathcal{I}_{a,inp}$	input domain of a (in $\mathcal{S}$), 141
$\mathcal{I}_{a,out}$	output domain of a (in $\mathcal{S}$), 141
$\mathcal{R}_a^{sipp}(\mathcal{S})$	predicate *is-sipp* for a in $\mathcal{S}$, 141
$\mathcal{R}_a^{wipp}(\mathcal{S})$	predicate *is-wipp* for a in $\mathcal{S}$, 141
$\mathcal{R}_a^{sopp}(\mathcal{S})$	predicate *is-sopp* for a in $\mathcal{S}$, 142
$\mathcal{R}_a^{wopp}(\mathcal{S})$	predicate *is-wopp* for a in $\mathcal{S}$, 142
$\mathcal{R}_a^{ms}(\mathcal{S})$	predicate *is-ms* for a in $\mathcal{S}$, 144
$\mathcal{R}_a^{sms}(\mathcal{S})$	predicate *is-sms* for a in $\mathcal{S}$, 144
$\mathcal{R}_a^{wms}(\mathcal{S})$	predicate *is-wms* for a in $\mathcal{S}$, 144
$\Sigma_{i,ext}$	set of external actions (of $\mathcal{C}_i$), 150
$\Sigma_{i,loc}$	set of locally-controlled actions (of $\mathcal{C}_i$), 150
j	element of $\mathcal{I}$, 150
J	subset of $\mathcal{I}$, 150
$\Sigma_{J,ext}$	set of external actions (of SUB_J), 150
$\Sigma_{J,loc}$	set of locally-controlled actions (of SUB_J), 150

6. Behavior of Team Automata

pREG	family of prefix-closed regular finitary languages, 164

REG	family of regular languages, 164
FIN	family of finite languages, 164
CA	$\{\,\mathbf{B}_{\mathcal{C}}^{\Sigma} \mid \mathcal{C}$ is a finite component automaton with alphabet $\Sigma\,\}$, 164
CA^{alph}	$\{\,\mathbf{B}_{\mathcal{C}}^{alph} \mid \mathcal{C}$ is a finite component automaton$\,\}$ (with $alph \in \{inp, out, int, ext, loc\}$), 165
$\mathcal{I}$	index set, 166
$\mathcal{C}_i$	component automaton, 166
Σ_i	set of actions (of $\mathcal{C}_i$), 166
$\mathcal{S}$	composable system, 166
$\mathcal{T}$	team automaton, 166
Σ	set of actions (of $\mathcal{T}$), 166
Θ	arbitrary alphabet disjoint from set Q of states (of $\mathcal{T}$), 166
j	element of $\mathcal{I}$, 166
$uAI_j(\mathcal{T})$	set of useful j-ai actions (of $\mathcal{T}$), 169
$\|\|$	shuffle, 183
$\|\|\|$	fair shuffle, 183
$\|\| d \|\|$	norm (of decomposition d), 198
$\|\|\|_{i\in[n]}$	n-ary fair shuffle, 205
$\|\|_{i\in[n]}$	n-ary shuffle, 205
$\|\|^{\Gamma}$	S-shuffle on Γ, 207
$\|\|\|^{\Gamma}$	fair S-shuffle on Γ, 207
$\mathrm{alph}(L)$	alphabet (of a language L), 208
${}_{\Sigma_1}\underline{\|\|}_{\Sigma_2}$	fS-shuffle w.r.t. Σ_1 and Σ_2, 208
${}_{\Sigma_1}\underline{\|\|\|}_{\Sigma_2}$	fair fS-shuffle w.r.t. Σ_1 and Σ_2, 208
${}_{\Sigma_1}\underline{\|\|}^{\Gamma}_{\Sigma_2}$	rS-shuffle on Γ w.r.t. Σ_1 and Σ_2, 209
${}_{\Sigma_1}\underline{\|\|\|}^{\Gamma}_{\Sigma_2}$	fair rS-shuffle on Γ w.r.t. Σ_1 and Σ_2, 209
$\|\|\|^{\Gamma}_{i\in[n]}$	n-ary fair S-shuffle on Γ, 227
$\|\|^{\Gamma}_{i\in[n]}$	n-ary S-shuffle on Γ, 227
$\underline{\|\|\|}_{\bigcup_{i\in[n]}\Sigma_i}$	n-ary fair fS-shuffle w.r.t. $\bigcup_{i\in[n]}\Sigma_i$, 228
$\underline{\|\|}_{\bigcup_{i\in[n]}\Sigma_i}$	n-ary fS-shuffle w.r.t. $\bigcup_{i\in[n]}\Sigma_i$, 228
$\underline{\|\|\|}^{\Gamma}_{\bigcup_{i\in[n]}\Sigma_i}$	n-ary fair rS-shuffle on Γ w.r.t. $\bigcup_{i\in[n]}\Sigma_i$, 228
$\underline{\|\|}^{\Gamma}_{\bigcup_{i\in[n]}\Sigma_i}$	n-ary rS-shuffle on Γ w.r.t. $\bigcup_{i\in[n]}\Sigma_i$, 228

7. Team Automata, I/O Automata, Petri Nets

$\mathcal{I}$	index set, 233
$\mathcal{C}_i$	component automaton, 233
Σ_i	set of actions (of $\mathcal{C}_i$), 233
$\mathcal{S}$	composable system, 233
$\mathcal{T}$	team automaton, 233

Σ	set of actions (of $\mathcal{T}$), 233
Σ_{ext}	set of external actions (of $\mathcal{T}$), 233
Σ_{loc}	set of locally-controlled actions (of $\mathcal{T}$), 233
Θ	arbitrary alphabet disjoint from set Q of states (of $\mathcal{T}$), 233
$\mathcal{S}$	compatible system, 237
$\mathcal{T}$	team I/O automaton, 239
IOCA	$\{\mathbf{B}_{\mathcal{C}}^{\Gamma} \mid \Gamma$ is an alphabet and $\mathcal{C}$ is a finite input-enabling component automaton with alphabet $\Gamma\}$, 240
IOCA^{alph}	$\{\mathbf{B}_{\mathcal{C}}^{alph} \mid \mathcal{C}$ is a finite input-enabling component automaton$\}$ (with $alph \in \{inp, out, int, ext, loc\}$), 240
$\Delta_a^v(\mathcal{S})$	complete vector transition space (of a in $\mathcal{S}$), 245
$\underline{a}$	vector action a, 245
$\mathcal{T}^v$	vector team automaton, 245
δ^v	set of labeled vector transitions (of $\mathcal{T}^v$), 245
$\delta_{\underline{a}}^v$	set of vector $\underline{a}$-transitions (of $\mathcal{T}^v$), 245
$SUB_J(\mathcal{T}^v)$	the subteam of $\mathcal{T}^v$ determined by J, 246
$\mathcal{T}_F^v$	the flattened version (of $\mathcal{T}^v$), 247
$tFree(\mathcal{T}^v)$	set of truly *free* actions (of $\mathcal{T}^v$), 250
$tAI(\mathcal{T}^v)$	set of truly *ai* actions (of $\mathcal{T}^v$), 250
$tSI(\mathcal{T}^v)$	set of truly *si* actions (of $\mathcal{T}^v$), 250
Λ	empty word vector, 252
$\mathrm{tot}\,(\{\Delta_j \mid j \in J\})$	total vector alphabet (over $\{\Delta_j \mid j \in J\}$), 252
Δ^u	subset of uniform vector letters of vector alphabet Δ, 252
$v \circ w$	component-wise concatenation (of two n-dimensional vector letters v and w), 252
coll	collapse of a sequence of vector letters into a word vector, 252
$\mathrm{und}(\mathcal{T}^v)$	underlying vector automaton (of $\mathcal{T}^v$), 253
$\mathbf{V}_{\mathcal{T}^v}$	finitary vector behavior (of $\mathcal{T}^v$), 253
$\mathbf{V}_{\mathcal{T}^v}^{\omega}$	infinitary vector behavior (of $\mathcal{T}^v$), 253
$\mathbf{V}_{\mathcal{T}^v}^{\infty}$	vector behavior (of $\mathcal{T}^v$), 253
$\mathcal{N}$	n-VLITN, 254
P	finite set of places (of $\mathcal{N}$), 254
T	finite set of events (of $\mathcal{N}$), 254
O	finite set of n integers, called tokens (of $\mathcal{N}$), 254
F	flow function (of $\mathcal{N}$), 254
V	vector alphabet of vector labels (of $\mathcal{N}$), 255
ℓ	event labeling homomorphism (of $\mathcal{N}$), 255
$\mathrm{use}\,(t)$	set of tokens used (by event t), 255
$\mathbf{M}_{\mathcal{N}}$	set of all markings of $\mathcal{N}$, 255
$\mu[t\rangle_{\mathcal{N}}$	enabled (an event t of $\mathcal{N}$ at a marking μ of $\mathcal{N}$), 256
$\mu[t\rangle_{\mathcal{N}}\nu$	fires (an event t of $\mathcal{N}$ from a marking μ of $\mathcal{N}$ to a marking ν of $\mathcal{N}$), 256

$\mu_0[t_1t_2\cdots t_m\rangle_{\mathcal{N}}$	firing sequence (of events $t_1, t_2, \ldots, t_m$) of $\mathcal{N}$ starting from μ_0, 256
$\mu_0[t_1t_2\cdots t_m\rangle_{\mathcal{N}}\mu_m$	firing sequence (of events $t_1, t_2, \ldots, t_m$) of $\mathcal{N}$ starting from μ_0 and leading to μ_m, 256
$\mu_0[t_1t_2\cdots\rangle_{\mathcal{N}}$	infinite firing sequence (of events $t_1, t_2, \ldots$) of $\mathcal{N}$ starting from μ_0, 256
$\mathcal{K}$	n-ITNC, 256
und($\mathcal{K}$)	underlying n-VLITN (of $\mathcal{K}$), 256
$\mathcal{M}_0$	set of initial markings (of $\mathcal{K}$), 256
$\mathcal{M}_f$	set of final markings (of $\mathcal{K}$), 256
$\mathbf{FS}_{\mathcal{K}}$	set of all firing sequences (of $\mathcal{K}$), 257
$\mathbf{M}_{\mathcal{K}}$	the set of all reachable markings (of $\mathcal{K}$), 257
$\mathbf{B}_{\mathcal{K}}$	behavior of $\mathcal{K}$, 257
$\mathbf{V}_{\mathcal{K}}$	vector behavior of $\mathcal{K}$, 257
carrier ($\underline{a}$)	carrier (of $\underline{a}$), 260
$PN(\mathcal{T}^v)$	ITNC obtained from $\mathcal{T}^v$, 261
$SUB_J(\mathcal{K})$	the subnet (of $\mathcal{K}$) determined by J, 270

8. Applying Team Automata

$\mathcal{I}$	index set, 278
$\mathcal{C}_i$	component automaton, 278
$\Sigma_{i,ext}$	set of external actions (of $\mathcal{C}_i$), 278
$\mathcal{S}$	composable system, 278
$\mathcal{T}$	team automaton, 278
Σ	set of actions (of $\mathcal{T}$), 278
Σ_{ext}	set of external actions (of $\mathcal{T}$), 278
$\mathcal{C}_H^{\Delta}$	the Δ-hiding version (of $\mathcal{C}$), 278
Σ_{com}	set of communicating actions (in $\mathcal{S}$), 279
$\boxed{\mathcal{T}}$	(communication) closed version (of $\mathcal{T}$), 279
$\mathcal{C}_N^h$	h-renamed version (of $\mathcal{C}$), 280

Index

Samenvatting

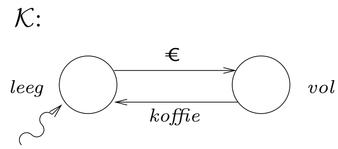
De Nederlandse titel van dit proefschrift is “Teamautomaten: een formele benadering van het modelleren van samenwerking tussen systeemcomponenten”. Dit proefschrift gaat dus over teamautomaten, een wiskundig model voor de beschrijving van het gedrag van reactieve en gedistribueerde systemen. Een reactief systeem is een systeem dat een voortdurende wisselwerking met zijn omgeving vereist — zoals een koffieautomaat. Een gedistribueerd systeem is een systeem dat uit verschillende en vaak fysiek verspreide componenten bestaat, maar dat middels een hechte samenwerking naar de buitenwereld toe wel degelijk de indruk geeft een samenhangend geheel te zijn — zoals het Internet. Een teamautomaat bestaat dan ook uit componenten die zelf ook weer automaten zijn en die op een bepaalde manier samenwerken.

Teamautomaten zijn in 1997 informeel geïntroduceerd door C.A. Ellis, met als belangrijkste motivatie het beschrijven en analyseren van groupwaresystemen. Dit zijn zowel software- als hardwaresystemen, die tot doel hebben groepen mensen in hun onderlinge samenwerking te ondersteunen — zoals email. Deze systemen zijn daardoor vaak reactief en gedistribueerd, maar bestaande modellen voor de beschrijving van zulk soort systemen werden door C.A. Ellis ontoereikend bevonden voor de specifieke vormen van samenwerking zoals die binnen groupwaresystemen plaatsvinden. Hierop besloot hij tot de introductie van teamautomaten als uitbreiding op de ‘Input/Output’-automaten die in 1987 door M.R. Tuttle en N. Lynch geïntroduceerd zijn.

De voornaamste doelen van dit proefschrift zijn (a) het wiskundig precies definiëren van teamautomaten, (b) het bepalen van de voorwaarden waaronder teamautomaten aan bepaalde eigenschappen voldoen, (c) het vergelijken van teamautomaten met verwante modellen uit de literatuur en (d) een aanzet geven tot het toepassen van teamautomaten in de praktijk.

Achtergrond

Automaten zijn toestandsovergangsmodellen die in de informatica veelvuldig gebruikt worden voor de beschrijving van het dynamische gedrag van (computer)systemen. Zo'n automaat bevindt zich op ieder moment in één bepaalde toestand. Wanneer er een verandering plaatsvindt in het systeem dat door de automaat beschreven wordt, dan wordt dit in de automaat weergegeven door de uitvoering van een actie die deze verandering symboliseert, met als gevolg dat de automaat zich in een nieuwe toestand begeeft. Naast computers komen er in het dagelijks leven nog vele andere systemen voor die goed door een automaat kunnen worden beschreven. Zoals het nu volgende voorbeeld laat zien kan hierbij gedacht worden aan koffieautomaten.



Bovenstaande automaat $\mathcal{K}$ geeft een hele simpele koffieautomaat weer. Deze koffieautomaat produceert een koffie na inwerping van een euro. De automaat $\mathcal{K}$ onderscheidt hiervoor twee mogelijke toestanden, *leeg* en *vol*, die aangeven of er wel of geen euro is ingeworpen. Initieel is er geen euro ingeworpen en *leeg* is dan ook de begintoestand van $\mathcal{K}$, wat is aangegeven middels een kronkelend pijltje. Het inwerpen van een euro wordt in $\mathcal{K}$ beschreven door het uitvoeren van de actie €, met als resultaat dat $\mathcal{K}$ zich in de toestand *vol* begeeft. Pas nu kan de koffieautomaat een koffie procuderen, wat in $\mathcal{K}$ wordt beschreven door het uitvoeren van de actie *koffie*. Dit procédé kan vervolgens eindeloos herhaald worden.

Het op een formele, wiskundige manier beschrijven en vervolgens analyseren van (computer)systemen vormt een belangrijk deelgebied van de informatica. Onderzoek in dit gebied heeft een groot aantal modellen en technieken voortgebracht, waaronder vele soorten automaten — inclusief teamautomaten. Het specifieke voordeel van het teamautomatenmodel is de flexibiliteit die het biedt met betrekking tot het beschrijven van verschillende soorten samenwerking tussen (componenten van) systemen.

Het Model

Een teamautomaat is een compositie van componentautomaten. Een componentautomaat is een automaat die drie soorten acties onderscheidt, namelijk invoer, uitvoer en interne acties. Invoer en uitvoer acties vormen tezamen de externe acties en zij kunnen worden gebruikt om allerlei vormen van samenwerking tussen de componentautomaten te modelleren. Welke vorm van samenwerking ook gekozen wordt, de resulterende teamautomaat zal technisch gezien weer een componentautomaat zijn. Dit maakt het mogelijk om teamautomaten te maken met teamautomaten als componenten.

De samenwerking tussen componentautomaten binnen een teamautomaat bestaat uit het simultaan uitvoeren (ook wel synchroniseren genoemd) van gemeenschappelijke acties. Gebaseerd op de gekozen vorm van samenwerking worden er in dit proefschrift verschillende soorten (synchronisaties van) acties gedefinieerd. Zo worden acties die nooit door meer dan één componentautomaat tegelijk worden uitgevoerd, vrij genoemd. Acties die altijd worden uitgevoerd als synchronisaties waaraan alle componentautomaten die de bewuste actie hebben meedoen, worden actie-onmisbaar ('action-indispensable') genoemd. Wanneer deze eis tot deelname wordt beperkt tot die componentautomaten die zich in een toestand bevinden waarin zij de bewuste actie kunnen uitvoeren, dan spreken we van toestand-onmisbare ('state-indispensable') acties. Door vervolgens rekening te houden met de verschillende rollen die acties kunnen hebben in componentautomaten, kunnen complexere vormen van synchronisatie worden benoemd. Zo worden in dit proefschrift 'peer-to-peer' synchronisaties — van acties van hetzelfde soort — en meester-slaaf synchronisaties — met uitvoer acties als meesters en invoer acties als slaven — gedefinieerd.

Resultaten

Hieronder volgt een handvol van de meest aansprekende resultaten van dit proefschrift. Deze hebben met elkaar gemeen hebben dat ze weinig of geen aanvullende uitleg behoeven om te kunnen worden gewaardeerd en laten bovendien zien dat de voornaamste doelen van dit proefschrift bereikt worden.

Zoals al eerder opgemerkt kan elke teamautomaat zelf weer gebruikt worden als component in de samenstelling van een nieuwe teamautomaat. In dit proefschrift wordt bewezen dat dit geïtereerd samenstellen van teamautomaten niet leidt tot een vergroting van het aantal mogelijkheden tot synchronisatie van de acties van de componentautomaten waaruit zij zijn samengesteld.

De verzameling van alle rijtjes van acties die door een teamautomaat vanuit een begintoestand achter elkaar kunnen worden uitgevoerd, vormen tezamen het gedrag (de taal) van deze teamautomaat. In dit proefschrift wordt bewezen dat een aantal van de in dit proefschrift gedefinieerde soorten synchronisatie zodanig is, dat het gedrag van elke teamautomaat die volgens zo'n soort synchronisatie is samengesteld bepaald kan worden zonder te weten hoe deze teamautomaat er precies uit ziet. Om deze vorm van compositionaliteit te bewijzen wordt een uitgebreide wiskundige theorie ontwikkeld over het op bepaalde manieren inéénrijgen ('to shuffle') van rijtjes van acties.

Bovenstaande resultaten met betrekking tot iteratie en compositionaliteit maken teamautomaten zeer geschikt om een abstracte hoog-niveau beschrijving van een systeem middels het stap voor stap vervangen van onderdelen van de huidige beschrijving door meer gedetailleerde beschrijvingen, te decomponeren in een meer concrete laag-niveau beschrijving. Dit is een in de informatica veelvuldig toegepaste techniek om complexe systemen toegankelijker te maken voor analysedoeleinden.

In de Introductie van dit proefschrift wordt kort bij overeenkomsten en verschillen tussen teamautomaten en verwante modellen stilgestaan. In een later hoofdstuk volgt een meer gedetailleerde vergelijking van teamautomaten met twee van deze modellen, namelijk het al eerder genoemde 'Input/Output'-automatenmodel en een model gebaseerd op Petri-netten. Er wordt bewezen dat 'safe Input/Output'-automaten (ook wel 'unfaire Input/Output'-automaten genoemd) ook formeel een deelmodel van teamautomaten zijn. Voor de vergelijking met Petri-netten wordt eerst overgestapt op een versie van teamautomaten genaamd vectorteamautomaten, waarin vectoren van acties in plaats van acties worden uitgevoerd. Vervolgens wordt bewezen dat een deelmodel van deze vectorteamautomaten vertaald kan worden in een Petri-netmodel genaamd 'Individual Token Net Controllers', dat in 1990 is geïntroduceerd door N.W. Keesmaat, H.C.M. Kleijn en G. Rozenberg.

De verscheidenheid aan vormen van samenwerking tussen de componenten van een teamautomaat maken het teamautomatenmodel bij uitstek geschikt voor het formeel beschrijven en analyseren van (componenten van) groupwaresystemen en hun interacties. Nadat C.A. Ellis dit al meteen bij de introductie van teamautomaten heeft geïllustreerd, wordt dit in dit proefschrift nogmaals duidelijk gemaakt door (onderdelen van) een gedistribueerde groupwarearchitectuur formeel te beschrijven als een teamautomaat. Tevens wordt een voorzichtig begin gemaakt met het analyseren van groupwaresystemen. Hieruit kan worden geconcludeerd dat het nuttig zou zijn om een computerprogramma (een 'tool') te ontwikkelen waarmee teamautomaten op een eenvoudige manier ontworpen kunnen worden en op bepaalde (gedrags)eigenschappen geanalyseerd kunnen worden. Dit verdient zonder twijfel nadere bestudering in de toekomst.

Curriculum Vitae

Maurice ter Beek werd op 7 oktober 1972 geboren te 's-Gravenhage en behaalde in 1990 zijn V.W.O.-diploma aan het Pallas College te Zoetermeer. Daarna begon hij met de studie informatica aan de Universiteit Leiden, alwaar hij als student-assistent ook vele werkcolleges heeft verzorgd. In 1995/1996 studeerde hij aan de Eötvös Loránd universiteit te Budapest met een beurs van het Hongaarse ministerie van onderwijs en cultuur. Gedurende die periode deed hij ook het onderzoek voor zijn afstudeerscriptie in het Computer and Automation Research Institute van de Hongaarse Akademie van Wetenschappen. Dit geschiedde onder de lokale begeleiding van Dr. E. Csuhaj-Varjú, vanuit Leiden gevolgd door Prof.dr. G. Rozenberg en Dr. H.C.M. Kleijn. Dit resulteerde in een afstudeerscriptie op het gebied van de theoretische informatica getiteld *Teams in grammar systems*. In 1996 studeerde hij — terug in Leiden — af in de informatica.

Vervolgens begon hij eind 1996 binnen de onderzoekschool IPA aan zijn promotie-onderzoek in de groep van Prof.dr. G. Rozenberg aan het LIACS te Leiden. Vanaf dat moment was de dagelijkse begeleiding steevast in handen van Dr. H.C.M. Kleijn. Gedurende bijna vijf jaar heeft hij, eerst als beurspromovendus, later als AIO, en tenslotte als docent, in Leiden hieraan gewerkt. In 1999 gebeurde dit grotendeels in Pisa, waar hij verbleef met een Erasmusbeurs van de EU. Begin 2001 verhuisde hij naar Pisa, maar in 2002 keerde hij voor een jaar terug naar Budapest om wederom aan het bovenstaande instituut — maar nu als ERCIM fellow — onderzoek te verrichten.

Vanaf januari 2003 is Maurice — terug in Pisa — als ERCIM fellow werkzaam in het Istituto di Scienza e Tecnologie dell'Informazione van het Consiglio Nazionale delle Ricerche te Pisa. Gedurende deze periode heeft hij zijn promotie-onderzoek afgerond, wat heeft geresulteerd in dit proefschrift.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms.* Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation.* Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search.* Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory.* Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems.* Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance.* Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction.* Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual.* Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits.* Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism.* Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference.* Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking.* Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics.* Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth.* Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context.* Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types.* Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics.* Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution.* Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing.* Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing.* Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering.* Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computa-*

tion. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search: Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On model checking the dynamics of object-based software: a foundational approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10